Department of
Computer Science

Technical Report

A Formal Framework and Extensions
for Function Approximation in Learning Classifier Systems

Jan Drugowitsch and Alwyn Barry

Technical Report 2006-01 **January 2006**
ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

A Formal Framework and Extensions for Function Approximation in Learning Classifier Systems

Jan Drugowitsch
Department of Computer Science
University of Bath, UK
J.Drugowitsch@bath.ac.uk

Alwyn M Barry
Department of Computer Science
University of Bath, UK
A.M.Barry@bath.ac.uk

January 2006

Abstract

In this paper we introduce part of a formal framework for Learning Classifier Systems (LCS) which, as a whole, aims at incorporating all components of LCS: function approximation, reinforcement learning and classifier replacement. The part introduced here concerns function approximation, and provides a formal problem definition, a formalisation of the LCS function approximation architecture, and a definition of the approximation aim. Additionally, we provide definitions of optimality and what conditions need to be fulfilled for a classifier to be optimal. Furthermore, as a demonstration of the usefulness of the framework, we derive commonly used algorithmic approaches that aim at reaching optimality from first principles, and introduce a new Kalman filter-based method that outperforms all currently implemented methods. How to mix classifiers to reach an overall approximation is simplified when compared to current LCS, and is justified by the Maximum Likelihood Estimate of a combination of all classifiers.

1 Introduction

Accuracy-based Learning Classifier Systems (LCS), which integrate function approximation, temporal difference learning and genetic algorithm driven search, are capable of the autonomous production of human-readable results that are the most compact generalised representation whilst also maintaining high predictive accuracy. Our previous work has demonstrated the competitiveness of this technique when applied to the task of data-mining [28, 17], work that has subsequently been extensively built upon (e.g. [5, 13, 3]). Other researchers have provided further demonstration of its power in many direct-reward environments, and have recently extended these results to the discovery of piecewise partially overlapping linear function approximations. However, our work has demonstrated that the approach has only limited success in other that relatively trivial delayed-reward tasks [4, 2]. These limitations have stimulated research to formulate partial models of the LCS (e.g. [8, 12, 31]). However, the recent theoretical developments have produced only piecemeal models that do not adequately capture the local dynamics of the components of the LCS (function approximation, temporal difference learning and rule discovery) and neglect the effect of interactions between the components that can invalidate the local models.

In order to make progress in understanding the inner workings of LCS we aim to produce a framework that permits the study of LCS as a whole rather than in pieces. The primary objectives of this work are to i) cover all LCS components within that framework to allow for systematic analysis of the components and their interaction, ii) design it flexibly to utilise it in developing extensions to current LCS methods, and iii) borrow notation and concepts from related fields to allow direct comparison and lower the barrier of translating new developments in other fields to LCS. Developing the framework was approached by splitting LCS into three components that relate to other Machine Learning techniques and are studied separately and in combination. Even though this paper only focusses on one of the components, this component is investigated with integration in mind.

Through recent developments (e.g. [35, 37, 23]), it became clear that in addition to the apparent evolutionary computation component, LCS also includes function approximation and reinforcement learning. The latter aims at learning a quality rating, called the *value*, for each environmental state, by interacting with the environment, receiving reward, and sensing features of the environmental states. The state values can be expressed as a function that maps features to those values, and will be approximated to overcome spatial and computational constraints. LCS use a special form of function

approximation where each classifier is only concerned with the value approximation for some subset of the state space. These values are integrated to form the overall approximation that guides the actions of the learning agent. Simultaneously, a Genetic Algorithm operates on the set of classifiers to find those who in combination allow for the most efficient approximation of the value function. Hence, the framework requires to unify the three components of LCS: function approximation (FA), reinforcement learning (RL), and classifier replacement.

This paper introduces the framework to study the function approximation component of LCS, and uses it to investigate and extend current algorithmic approaches. Wilson was probably the first to directly link LCS with function approximation [36]. Since then, the function approximation capabilities were improved by allowing for more complex approximation architectures [37, 24], and by introducing different algorithmic approaches to approximation [25]. Their approach, however, was ad-hoc rather than introducing new algorithms in a bottom-up approach from first principles.

To study the function approximation architecture that is facilitated in LCS, we will take some basic assumptions to allow relating our investigations to existing literature in function approximation and adaptive filter theory:

- The set of classifiers is fixed. This is equivalent to keeping the set of states that a classifier represents time-invariant.

- The value function is time-invariant. That assumption is fulfilled in direct-reward environments, but requires modification when investigating the interaction between function approximation and reinforcement learning (see [15]).

- The states are observed according to a time-invariant probability distribution. This is usually the case in the long run.

Some of the assumptions will have to be removed once the interaction between different components of LCS will be studied. However, even when only considered in combination with replacing classifiers, it can stand on its own as a method of adaptive function approximation (e.g. [36]) and for use in data-mining (e.g. [5]).

The function approximation-part of the framework is introduced in Section 2, together with the aim of function approximation in LCSs. Its development is based on Bertsekas and Tsitsiklis' work that investigates the interaction between reinforcement learning and function approximation [6], and among other work in reinforcement learning on the notation of Sutton and Barto's introductory text [29]. As such, it should be easily accessible for anyone working in that field. First use of that framework is demonstrated in Section 2.4, where we discuss partial matching and the likely consequences of a different form of approximation aim.

Section 3 is exclusively concerned with single classifiers and what it means for them to have an optimal approximation. That section is particularly important as it not only states the definite approximation goal for single classifiers, but also gives the conditions under which this optimality is reached. This is done for the case when the full function is known, and for the sample-based case which aims at updating the approximation with every additional observation of a function value. Additionally, we demonstrate for a finite state space that the latter case is an appropriate approximation to approximating the full function at once. Furthermore, we present a different point-of-view on the meaning of optimality for the case of state spaces of countable size by relating it to geometric concepts of linear algebra.

As a logical consequence of the previous section, Section 4 follows on by discussing different algorithms that aim at implementing a sample-based approach towards optimality as outlined before. Additionally, as a further demonstration of the usefulness of the framework, we utilise it in that section to develop a new and most likely superior algorithmic approach that is based on the Kalman filter [21], and relates to Recursive Least Squares. A short experimental section follows that demonstrates this superiority.

Last, but not least, in Section 5 we discuss how to best integrate the approximations of the classifiers into one approximation over the whole state space. That method is based on modelling the value approximation of a classifier by a normal distribution and using the Maximum Likelihood Estimate to calculate the most likely overall value. The section closes by investigating a mixing parameter and giving recommendations on how to set it.

2 The Framework

In this section we define the framework which will be used to study how Learning Classifier Systems use their classifiers to approximate functions. We will first give a general problem formulation, followed by the overall aim, and how this aim is approached by Learning Classifier Systems. In addition, we will present a short discussion on i) partial rather than exclusive (binary) matching and, ii) why LCS function approximation architectures that do *not* emphasise the independence of classifiers w.r.t. approximation might cause problems when used in combination with classifier replacement.

2.1 Problem Formulation and Aim

Let $V : S \to \mathbb{R}$ be the function we want to approximate[1]. The function's domain S, called the *state space*, is either a countable set or an uncountable set, which we will map onto $\mathbb{N}$ or $\mathbb{R}$ respectively[2].

The function V is not directly observable but will be sampled or observed with a given time-invariant sampling distribution. Let $\pi : S \to [0, 1]$ define the sampling distribution[3], giving the probability of sampling state i by $\pi(i)$. The function is sampled in discrete time-steps $t = 0, 1, \dots$, giving the sequence of states $\{i_0, i_1, \dots\}$, the sequence of function values $\{V(i_0), V(i_1), \dots\}$, and the sequence of approximations $\{\tilde{V}_0, \tilde{V}_1, \dots\}$, where $\tilde{V}_t : S \to \mathbb{R}$ is the approximation after observing $V(i_t)$[4].

In addition to observing the sequence of function values, we simultaneously observe a set of features of the current state. These features are formed through a set of basis functions $\{\phi_l : S \to \mathbb{R}\}_{l=1,\dots,L}$, where function $\phi_l(i)$ returns the lth feature of state i. In combination, they form the *feature vector* $\phi : S \to \mathbb{R}^L$, returning the features of state i as a vector $\phi(i) = (\phi_1(i), \dots, \phi_L(i))'$. Note that this as well as all other vectors in this paper is a column vector, unless otherwise stated, as indicated by the transpose $\cdot'$.

The selection of features is either left to the designer of the system or depends on the set of sensors that are available. Even though the quality of the approximation is highly dependent on the available features, we will not discuss good heuristics for selecting those features, and the interested reader is referred to standard literature about linear function approximation architectures, and the large body of existing work on feature selection.

The aim of the function approximator is to minimise the mean-squared error (MSE) given by

$$\int_S \pi(i)(V(i) - \tilde{V}(i))^2 \mathrm{d}i,$$

where $\tilde{V} : S \to \mathbb{R}$ is the approximation of V. For a countable set S the integral can be replaced by a sum. The square error is weighted by the sampling distribution, as it emphasises the error of states that are visited most frequently, and as it naturally represents the approximation target of most standard approximation algorithms.

2.2 LCS Function Approximation Architecture

An LCS utilises a finite set of K classifiers to approximate the function V. We will enumerate the classifiers with $1, \dots, K$, and denote a classifier parameter of classifier k by the subscript $\cdot_k$.

Each classifier k *matches* a particular subset $S_k \subseteq S$ of the state space S, which we will call the *match set*[5]. The aim of a classifier is to approximate the function V over the classifier's match set S_k. To ease notation, we will utilise the indicator function $I_{S_k} : S \to \{0, 1\}$, which is defined as

$$I_{S_k}(i) = \begin{cases} 1 & \text{if } i \in S_k, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Hence, $I_{S_k}(i)$ returns only 1 if classifier k matches state i.

To avoid introducing additional complexity, we will restrict ourselves to linear approximation architectures. Nevertheless, it is possible to use LCSs with non-linear approximation, but that might introduce local optima (see e.g. [19]) and will significantly complicate analysis. Linear architectures

[1] The symbol V is used with foresight to value function approximation in delayed-reward environments.

[2] Please note that all of the following is appropriate for the case of a countable state space, but might require some addition technical conditions for an uncountable state space, which are omitted to ease readability.

[3] If the dynamics of the sampling is determined by a Markov Chain, as the case for application in delayed-reward environments, this sampling distribution can be thought of as the steady-state distribution of the Markov Chain.

[4] In general, any symbol with subscript $\cdot_t$ indicates the time dependency of its meaning and refers to its state after observing the properties of state i_t.

[5] This does not conform to LCS jargon, where a match set is the set of classifiers that match a particular state.

are characterised by the independence between the approximation parameters and the features of the function to approximate. In that case, the approximation parameters are the adaptable values that modify the shape of the approximation, and in our case are given by the parameter vector $w_k \in \mathbb{R}^L$, also called the *weight vector*, of classifier k. For any matched state i, the dot product of the classifier's weight vector and the feature vector $\phi(i)$ of that state gives the approximation $\tilde{V}_k(i)$ by classifier k, that is

$$\tilde{V}_k(i) = w_k' \phi(i). \tag{2}$$

Our approximation goal is for each classifier f_k to minimise the mean-squared error $f_k : \mathbb{R}^L \to \mathbb{R}$ over its match set S_k, that is, we want to minimise

$$\begin{aligned} f_k(w_k) &= \frac{1}{\int_{S_k} \pi(i) \mathrm{d}i} \int_{S_k} \pi(i)(V(i) - \tilde{V}_k(i))^2 \mathrm{d}i \\ &= \frac{1}{\int_S I_{S_k}(i)\pi(i) \mathrm{d}i} \int_S I_{S_k}(i)\pi(i)(V(i) - \tilde{V}_k(i))^2 \mathrm{d}i \\ &= \frac{1}{\mathbb{E}(I_{S_k})} \mathbb{E}\left(I_{S_k} (V - \tilde{V}_k)^2 \right) \\ &= \mathbb{E}_k \left((V - \tilde{V}_k)^2 \right), \end{aligned} \tag{3}$$

where $\mathbb{E}$ is the expectation operator. The whole term is scaled by the inverse of the probability of observing any of the matching states to make the errors comparable between different classifiers. To ease notation, we define

$$\mathbb{E}_k(X) = \frac{1}{\mathbb{E}(I_{S_k})} \mathbb{E}(I_{S_k} X),$$

giving the normalised expectation over the states that classifier k matches. As before, for a countable set S the integral of Eq. (3) can be replaced by a sum.

As the function is usually not fully observable, the mean-squared error cannot be evaluated when performing sample-based approximation. Therefore, we will define an approximation $\varepsilon_{k,t}$ to the mean-squared error f_k at time t, which we define by

$$\varepsilon_{k,t} = \frac{1}{c_{k,t} - 1} \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w_{k,t}' \phi(i_m))^2, \tag{4}$$

where $c_{k,t}$ is the *match count*[6] of classifier k at time t, giving the number of observed states until time t that are in the match set S_k, that is

$$c_{k,t} = \sum_{m=0}^{t} I_{S_k}(i_m).$$

Note that the sequence of states $\{i_m\}$ is determined by the sampling distribution π, which will automatically introduce a weighting by this distribution in Eq. (4), as is required to approximate Eq. (3).

A feature of LCSs is to track the utility of a classifier at the same time as performing the approximation. For accuracy-based classifiers, the utility is defined as some inverse of the approximation of the mean-squared error $\varepsilon_{k,t}$. Thus, when discussing possible algorithms for classifier parameter approximation in LCSs, we need to consider tracking the approximation error in addition to providing a useful approximation.

As one classifier possibly only matches a subset of the state space, i.e. $S_k \subset S$, we need to integrate the approximation of all classifiers to recover an approximation over the whole domain of V. Let $\psi_k : S \to [0, 1]$ be the mixing weight of classifier k for state i, where $\sum_{k=1}^{K} \psi_k(i) = 1$ for any $i \in S$, and $\psi_k(i) = 0$ for all $i \notin S_k$. We can then form the overall approximation $\tilde{V}$ of V by

$$\tilde{V}(i) = \sum_{k=1}^{K} \psi_k(i) \tilde{V}_k(i) = \sum_{k=1}^{K} \psi_k(i) w_k' \phi(i). \tag{5}$$

As all mixing weights for one state sum up to 1, the overall approximation is always bounded by the highest and lowest approximation for that state. Additionally, classifiers that do not match state i do not contribute to the approximation of that state, due to the condition $\psi_k(i) = 0$ for all $i \notin S_k$. When we come to examine specific algorithms in Section 4, we will only deal with a single classifier, but we will discuss mixing strategies again in Section 5.

[6] In traditional LCS jargon, the match count is called the *experience* of a classifier.

Example 2.1 (Common Linear Approximation Architectures in LCS). As also employed in Wilson's original XCS [35], the most commonly used feature choice in LCSs is $\phi(i) = 1$ for all $i \in S$. As the classifier weight vector w_k has the same number of elements as the feature vector, we only have one element $w_k(1)$, which directly represents the approximated function value $V_k(i) = w_k(1)$, for all $i \in S_k$. This scalar is called the *prediction* in LCS jargon. In minimising the mean-squared error f_k (Eq. (3)), the classifier weight averages the function values over the states that the classifier matches, weighted by the sampling distribution. For that reason we will call such classifiers *averaging classifiers*.

Recently, the power of LCSs was increased by introducing more complex feature vectors, such as in [37, 38, 24]. Initially, a feature vector $\phi(i) = (1, i)'$ for all $i \in S$ was used. This allows a classifier to approximate straight lines by $V_k(i) = w_k(1) + iw_k(2)$, where $w_k(1)$ determines the bias and $w_k(2)$ gives the slope. Lanzi et al. [24] extended this concept by using feature vectors $\phi(i) = (1, i, i^2)'$ and $\phi(i) = (1, i, i^2, i^3)'$ to represent quadratic and cubic functions. Naturally, the choice of feature vectors is not restricted to nth-order polynomials, but can also include radial-basis functions (RBFs) or any other function, as long as they can be formulated as defined by Eq. (2).

2.3 Matrix Notation

In case of a countable state space, we will introduce the following matrix notation: The function V can be expressed though a column vector of size N with the elements $V = (V(1), \dots, V(N))'$. The features of all the states are combined into the $N \times L$ feature matrix Φ by representing the features of i in the ith row of the matrix, that is

$$\Phi = \begin{pmatrix} - & \phi(1)' & - \\ & \dots & \\ - & \phi(N)' & - \end{pmatrix}.$$

The state distribution is captured by the diagonal $N \times N$ matrix D with its diagonal entries $\pi(1), \dots, \pi(N)$.

Matching of classifier k is expressed through the diagonal $N \times N$ matching matrix I_{S_k} with the diagonal entries $I_{S_k}(1), \dots, I_{S_k}(N)$. Note that due to binary matching,

$$(I_{S_k})^a = I_{S_k} \qquad \forall a \in \mathbb{R}_{\neq 0}.$$

Let D_k be the $N \times N$ diagonal matrix, defined by $D_k = I_{S_k} D$, that contains the sampling probabilities only for the states that classifier k matches. The approximation of classifier k can be expressed by the vector $\tilde{V}_k$ of size N, and is given by $\tilde{V}_k = \Phi w_k$. The mixing weight is given by the diagonal $N \times N$ mixing matrix Ψ_k with diagonal entries $\psi_k(1), \dots, \psi_k(N)$. Note that due to the nature of ψ_k, $\sum_{k=1}^K \Psi_k = I$. The overall approximation of V is given by the N-sized vector $\tilde{V}$, which we can now define by

$$\tilde{V} = \sum_{k=1}^K \Psi_k \tilde{V}_k = \sum_{k=1}^K \Psi_k \Phi w_k.$$

The m-norm of vector $z \in \mathbb{R}^p$ is defined as $\|z\|_m = \left(\sum_{q=1}^p z(q)^m\right)^{\frac{1}{m}}$. The Euclidean norm is the 2-norm $\|\cdot\|_2$, which we will simply denote by $\|\cdot\|$. The weighted norm $\|z\|_A$, weighted by the diagonal elements $a(0), \dots, a(M)$ of the diagonal $M \times M$ matrix A is defined as $\|z\|_A = \sqrt{\sum_{q=1}^M a(q) z(q)^2}$. The induced m-norm of some matrix B is defined by $\|B\|_m = \max_{\|z\|_m = 1} \|Bz\|_m$.

2.4 Further Issues

2.4.1 Non-Binary Matching

Non-binary matching means using a matching-representing function (in our case I_{S_k}) that returns values of range $[0, 1]$, allowing for matching to a degree. Such matching was already experimented with in strength-based LCSs, and is also proposed for accuracy-based LCSs (e.g. [11]). Nevertheless, to our knowledge there exists no proper comparison between binary and non-binary matching. Butz only mentions that "Preliminary experiments in that respect [...] did not yield any further improvement in performance" [11].

By looking at Eq. (3) we can see that the indicator function I_{S_k} acts as a multiplier to the sampling distribution and can hence be interpreted as a modification of it. From the point-of-view of a classifier, there is no difference between not matching and not visiting a state, as $I_{S_k}(i)\pi(i) = 0$ for all $i \notin S_k$. Equally, states that are matched are visited according to their sampling distribution. Note that for

$I_{S_k}(i)\pi(i)$ to be the probabilities of a classifier-centric state distribution, we need to normalise it by the constant $\mathbb{E}(I_{S_k})^{-1}$, as it otherwise might violate $\int_{S_k} \pi(i)\mathrm{d}i = 1$.

When applying the above interpretation to non-binary matching, I_{S_k} can take values between 0 and 1 in addition to $\{0, 1\}$. Hence, rather than just choosing which states are observed and which are not, a reduced matching value has the effect of reducing the sampling probability of that state. Thus, when minimising Eq. (3) the approximation error of states with a lower matching value has reduced influence on our minimisation goal. As classifiers still approximate independently, we have no reason to believe that non-binary matching might not work, as long the a lower degree of matching is also considered in the algorithmic implementation of the mean-squared error after Eq. (4). However, due to the lack of experimental evidence, and to keep further analysis simple, we will only consider the case of binary matching, as given by Eq. (1).

2.4.2 Arguments against Aggregating Classifier Approximations

Recently, Wada et al. [31] have introduced a modified LCS, similar to Kovac's SXCS [23] and related to Wilson's ZCS [34], that removes the independence of function approximation between different classifiers. In our current framework, each classifier aims at finding the best approximation of its matching part of the function V, independent of the approximation of other classifiers. Another approach is to not minimise the error of each classifier separately, as given by Eq. (3), but to minimise the error of the overall approximation, that is to minimise

$$\int_S \pi(i) \left(V(i) - \sum_{k=1}^{K} I_{S_k}(i) w_k' \phi(i) \right)^2 \mathrm{d}i.$$

In that case, the function is approximated by

$$\tilde{V}(i) = \sum_{k=1}^{K} I_{S_k}(i) w_k' \phi(i),$$

without considering different mixing weights for different classifiers, in contrast to Eq. (5). Specifically, the approximation of the different classifiers is aggregated rather than averaged. Such a measure linearises classifier mixing and eases analysis by relating it to reinforcement learning with linear function approximation.

Let us now consider the consequences of replacing classifiers in the population[7]. In the case of aggregating the classifier approximations, removing one classifier from the population will also remove its contribution to the approximation of all the states that it matched. Therefore, the approximation for all those states will change, resulting in a deviation from the minimal error. Hence, caused by the aggregated approximation, the classifiers matching those states need to correct their approximations to again return to the optimal approximation for the new population. The necessary change might be significant and can cause severe temporary performance drops. Additionally, estimating the quality of a single classifier becomes harder, as the error of an approximation cannot be assigned to a single classifier but only to all classifiers that match that state. As a result, it becomes hard for accuracy-based LCSs with aggregating classifiers to judge the quality of a classifier.

Using averaging rather than aggregation removes the interdependence between the different classifiers. Replacing classifiers in the population does not influence the optimal approximation of other classifiers and allows for more stability in the overall approximation. The error of a single classifier can be easily judged by the mean-squared error over its matching states, as given by Eq. (3).

For all the above reasons we believe that aggregating classifier approximations will introduce more problems than it solves. An analysis that uses such an architecture as its basis deviates too much from the spirit of current LCSs to be of much support for future development. Therefore, we will only concentrate on the case of averaged classifier approximation, as outlined in the description of our framework.

3 Optimality

This section deals with what it means for an approximation performed by a single classifier to be *optimal*. In addition, the case of a state space of countable size is discussed and its relation to projection of the function V into different subspaces is highlighted.

[7] Even though we currently only consider constant populations, the former analysis was done with classifier replacement in mind.

3.1 Optimal Approximation and Minimal Error

3.1.1 Optimality given Full Knowledge of the Function

Let us consider classifier k and assume that the function V is completely known. Then our approximation aim is to minimise the mean-squared error as defined by Eq. (3). In addition to providing a good approximation, a good classifier in accuracy-based LCSs also needs to provide some measure of quality of itself. Naturally, the approximation error gives a good idea how well a classifier is able to reflect the function V over its match set. As we aim at minimising this error, we will use the minimal approximation error as a measure of quality of a classifier.

Using the definition of the mean-squared error according to Eq. (3), and the linear approximation architecture of classifiers, as given by Eq. (2), we can state the following:

Theorem 3.1. *Given that the function $V : S \to \mathbb{R}$ is fully known, the approximation of classifier k is optimal (i.e. the approximation error is minimal) if the features are orthogonal to the approximation error, that is if*

$$\int_{S_k} \pi(i)\phi(i)(V(i) - w_k'\phi(i))\mathrm{d}i = 0$$

holds, which can be rewritten as

$$\mathbb{E}_k(\phi\phi')w_k = \mathbb{E}_k(V\phi).$$

In that case, the mean-squared error of that classifier is

$$\frac{1}{\int_{S_k} \pi(i)\mathrm{d}i}\left(\int_{S_k} \pi(i)V(i)^2\mathrm{d}i - \int_{S_k} \pi(i)\tilde{V}_k(i)^2\mathrm{d}i\right),$$

which is the minimal error that can be achieved with that classifier, and can be rewritten as

$$\mathbb{E}_k(V^2) - \mathbb{E}_k(\tilde{V}_k^2).$$

Proof. Combining Eq. (3) and Eq. (2), and omitting the constant term $\mathbb{E}(I_{S_k})^{-1}$, gives

$$\int_{S_k} \pi(i)(V(i) - w_k'\phi(i))^2\mathrm{d}i$$

as the minimisation goal for classifier k. The optimality condition can be found by taking the first derivative of above w.r.t. w_k and setting it to zero. Given that a minimum exists, the resulting minimum is unique, as the above is a convex function of w_k. Its form $\mathbb{E}_k(\phi\phi')w_k = \mathbb{E}_k(V\phi)$ is a rewrite of the optimality condition in the form

$$\left(\pi(i)\int_{S_k} \phi(i)\phi(i)'\mathrm{d}i\right)w_k = \int_{S_k} \pi(i)V(i)\phi(i)\mathrm{d}i,$$

which is then multiplied by $\mathbb{E}(I_{S_k})^{-1}$ on both sides.

The simplified error term can be derived by using $V(i) = w_k'\phi(i) + (V(i) - w_k'\phi(i))$ and the optimality condition to get

$$\begin{aligned}\int_{S_k} \pi(i)V(i)^2 &= \int_{S_k} \pi(i)(w_k'\phi(i))^2\mathrm{d}i \\ &\quad + 2w_k'\int_{S_k} \pi(i)\phi(i)(V(i) - w_k'\phi(i))\mathrm{d}i \\ &\quad + \int_{S_k} \pi(i)(V(i) - w_k'\phi(i))^2\mathrm{d}i \\ &= \int_{S_k} \pi(i)\tilde{V}_k(i)^2\mathrm{d}i + \int_{S_k} \pi(i)(V(i) - \tilde{V}_k(i))^2\mathrm{d}i.\end{aligned}$$

The second integral of the right-hand side is the mean-squared error scaled by $\mathbb{E}(I_{S_k})$, which leads straight to the minimal error expression. □

To attach some more intuitive understanding, we will call the matrix $\mathbb{E}_k(\phi\phi')$ the feature vector correlation matrix, and $\mathbb{E}_k(V\phi)$ the function-feature correlation matrix[8].

[8] Technically, calling these terms *correlation matrices* is a slight misuse of the term, as they might be shifted by a certain bias. Still, as they express some form of correlation within the pairs $\phi\phi'$ and $V\phi$, we feel that calling them correlation matrices is appropriate.

3.1.2 Optimality by Sampling

Usually, we cannot observe all of the function V at once. Instead, a sequence of observations $\{V(i_0), V(i_1), \dots\}$ is used to perform its approximation. Hence, at every time t we aim to minimise the approximated mean-squared error $\varepsilon_{k,t}$, as given by Eq. (4). However, also as above, we can derive optimality conditions at each time t, and get the following result, of which the optimality condition is known as the *Principle of Orthogonality* (see e.g. [20]):

Theorem 3.2 (Principle of Orthogonality). *The approximation of function V by classifier k is optimal (i.e. the approximation error is minimal) if the sequence of feature vectors $\{\phi(i_0), \phi(i_1), \dots\}$ is orthogonal to the sequence of approximation errors for the matching states $i_m \in S_k$. That is, at time t,*

$$\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)(V(i_m) - \tilde{V}_{k,t}(i_m)) = 0 \tag{6}$$

has to hold, where $\tilde{V}_{k,t}(i_m) = w'_{k,t}\phi(i_m)$, and $w_{k,t}$ is the classifier's weight vector at time t.

In that case, the approximated mean-squared error of classifier k is

$$\varepsilon_{k,t} = \frac{1}{c_{k,t} - 1} \sum_{m=0}^{t} I_{S_k}(i_m) \left(V(i_m)^2 - \tilde{V}_{k,t}(i_m)^2 \right),$$

which is equally the minimum approximation error at time t.

Proof. For a constant time t, the match count $c_{k,t}$ is also a constant and is therefore irrelevant for minimisation. Thus, it is sufficient to minimise

$$\sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{k,t}\phi(i_m))^2,$$

which is a convex function of $w_{k,t}$. Therefore, we can derive above optimality condition by taking the first derivative and setting it to zero.

We will derive the mean-squared error by using its definition (Eq. (4)), the optimality condition, and $V(i) = w'_{k,t}\phi(i) + (V(i) - w'_{k,t}\phi(i))$, to get

$$\begin{aligned}
\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)^2 &= \sum_{m=0}^{t} I_{S_k}(i_m)(w'_{k,t}\phi(i_m))^2 \\
&\quad + 2w'_{k,t} \sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)(V(i_m) - w'_{k,t}\phi(i_m)) \\
&\quad + \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{k,t}\phi(i_m))^2 \\
&= \sum_{m=0}^{t} I_{S_k}(i_m)(w'_{k,t}\phi(i_m))^2 + \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - \tilde{V}_{k,t}(i))^2.
\end{aligned}$$

The second result follows from observing that, multiplied by $(c_{k,t} - 1)^{-1}$, the second sum on the right-hand side is the mean-squared error at time t. □

If we pre-multiply the optimality condition from the Principle of Orthogonality by $w'_{k,t}$, we get

$$\sum_{m=0}^{t} I_{S_k}(i_m)w'_{k,t}\phi(i_m)(V(i_m) - w'_{k,t}\phi(i_m)) = 0.$$

Together with Eq. (2) that gives:

Corollary 3.3 (Corollary to the Principle of Orthogonality). *The approximation of function V by classifier k is optimal (i.e. the approximation error is minimal) if the sequence of the approximations $\{\tilde{V}_{k,t}(i_0), \tilde{V}_{k,t}(i_1), \dots\}$ is orthogonal to the sequence of approximation errors for the matching states $i_m \in S_k$. That is, at time t*

$$\sum_{m=0}^{t} I_{S_k}(i_m)\tilde{V}_{k,t}(i_m)(V(i_m) - w'_{k,t}\phi(i_m)) = 0$$

has to hold, where $w_{k,t}$ is the classifier's weight vector at time t, and $\tilde{V}_{k,t}$ is its approximation, given by $\tilde{V}_{k,t}(i) = w'_{k,t}\phi(i)$ for state i.

Example 3.1 (Optimal Approximation for Averaging Classifiers). Let us consider an averaging classifier k, with a feature vector of $\phi(i) = (1)$ for all $i \in S$, in which case by Eq. (2) $\tilde{V}_k(i) = w_k(1)$ for all $i \in S_k$. Then, by solving Theorem 3.1 for w_k, the optimal approximation given a known function V is

$$w_k(1) = \mathbb{E}_k(\phi\phi')^{-1}\mathbb{E}_k(V\phi) = \mathbb{E}_k(V),$$

which is the normalised expectation over function V for all matched states. In that case, the mean-squared error is

$$\begin{aligned}\mathbb{E}_k(V^2) - \mathbb{E}_k(\tilde{V}_k^2) &= \mathbb{E}_k(V^2) - \mathbb{E}_k\left(\mathbb{E}_k(V)^2\right)\\ &= \mathbb{E}_k(V^2) - \mathbb{E}_k(V)^2\\ &= \mathrm{var}_k(V),\end{aligned}$$

where var_k stands for the normalised variance over all matches states, that is

$$\mathrm{var}_k(X) = \frac{1}{\mathbb{E}(I_{S_k})}\mathrm{var}(I_{S_k}X).$$

This demonstrates that the mean-squared error is hence the normalised variance of the function values over the matched states.

For the sample-based approach, let us consider the state sequence $\{i_0, \dots i_t\}$ at time t. Then, if we substitute Eq. (2) in Theorem 3.2 and solve for $w_{k,t}$, the optimal approximation at time t is

$$\begin{aligned}w_{k,t}(1) &= \left(\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)'\right)^{-1}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m)\\ &= \frac{1}{c_{k,t}}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m),\end{aligned}$$

which is the average of the function values of all matched states in the state sequence. The approximated mean-squared error becomes

$$\varepsilon_{k,t} = \frac{1}{c_{k,t}-1}\sum_{m=0}^{t} I_{S_k}(i_m)\left(V(i_m)^2 - \left(\frac{1}{c_{k,t}}\sum_{p=0}^{t} I_{S_k}(i_p)V(i_p)\right)^2\right),$$

which is the sample variance of the function value of all matched states up until time t.

3.2 Suitability of the MSE Approximation

Previously, we have stated Eq. (4) as a sample-based approximation to Eq. (3) without showing the suitability of that approximation. In this section we will develop a short proof for finite state spaces that demonstrates the appropriateness of the approximation. For the continuous case, the interested reader is referred to [16].

The proof is based on the law of large numbers and uses the following result:

Lemma 3.4. *Let $\kappa_t(i)$ be the number of times that state i was observed up to and including time t, and $\pi(i) > 0$ for all $i \in S$. Then, for some fixed $i \in S$,*

$$\lim_{t\to\infty}\frac{1}{t+1}\kappa_t(i) = \pi(i), \text{ with probability 1.}$$

Proof. Let $X_0, X_1, \dots$ be identically distributed Bernoulli random variables with $\mathbb{P}(X_0 = 1) = \pi(i)$ for some fixed $i \in S$. As we are sampling according to the probabilities $\pi(1), \pi(2), \dots$, let $\kappa_t(i) = \sum_{m=0}^{t} X_m$. Then, by the strong law of large numbers (e.g. [18, Ch. 7.5]),

$$\frac{1}{t+1}\kappa_t(i) = \frac{1}{t+1}\sum_{m=0}^{t} X_m \to \pi(i), \text{ as } t \to \infty, \text{ with probablity 1.}$$

□

This Lemma leads directly to the next result:

Theorem 3.5. *Let S be a finite state space with $\pi(i) > 0$ for all $i \in S$. Then, given that $w_{k,t}$ converges to w_k, the approximation of the mean squared error $\varepsilon_{k,t}$, as given by Eq. (4), converges to the mean-squared error f_k, given by Eq. (3), as $t \to \infty$, with probability 1.*

Proof. Let $\kappa_t(i)$ denote the number of times that state i was observed up to and including time t, giving

$$\begin{aligned}\varepsilon_{k,t} &= \left(\sum_{m=0}^{t} I_{S_k}(i_m) - 1\right)^{-1} \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{k,t}\phi(i_m))^2 \\ &= \left(\frac{1}{t+1}\sum_{i\in S} I_{S_k}(i)\kappa_t(i) - \frac{1}{t+1}\right)^{-1} \frac{1}{t+1}\sum_{i\in S} I_{S_k}(i)\kappa_t(i)(V(i) - w'_{k,t}\phi(i))^2\end{aligned}$$

By Lemma 3.4 we know that

$$\lim_{t\to\infty}\left(\frac{1}{t+1}\sum_{i\in S} I_{S_k}(i)\kappa_t(i) - \frac{1}{1+t}\right)^{-1} = \left(\sum_{i\in S} I_{S_k}(i)\pi(i)\right)^{-1},$$

and, as $w_{k,t} \to w_k$,

$$\lim_{t\to\infty}\frac{1}{t+1}\sum_{i\in S} I_{S_k}(i)\kappa_t(i)(V(i) - w'_{k,t}\phi(i))^2 = \sum_{i\in S} I_{S_k}(i)\pi(i)(V(i) - w'_k\phi(i))^2.$$

Hence,

$$\begin{aligned}\lim_{t\to\infty}\varepsilon_{k,t} &= \left(\sum_{i\in S} I_{S_k}(i)\pi(i)\right)^{-1} \sum_{i\in S} I_{S_k}(i)\pi(i)(V(i) - w'_k\phi(i))^2 \\ &= \mathbb{E}(I_{S_k})^{-1}\mathbb{E}(I_{S_k}(V - w'_k\phi)^2) \\ &= f_k(w_k).\end{aligned}$$

□

Therefore, the approximated mean-squared error is a suitable approximation to the mean-squared error.

3.3 A Different View for Countable State Spaces

In this section we will use the matrix notation, as defined in Section 2.3, to develop a different view of optimal approximations in terms of projections into different subspaces. Some of the concepts of this section are borrowed from [30].

Let us first rewrite the mean-squared error f_k in matrix notation:

$$f_k(w_k) = \frac{1}{\mathrm{tr}(D_k)}\|V - \Phi w_k\|^2_{D_k},$$

where $\|\cdot\|_{D_k}$ is the vector norm weighted by the sampling distribution of the matched states, and $\mathrm{tr}(D_k)$ is the trace of matrix D_k, giving the sum of the sampling probabilities of the matched states. Hence, the error is proportional to the weighted distance between the vector V and its approximation $\tilde{V}_k = \Phi w_k$. By the definition of the weighted norm, we can observe that, for some vector $z \in \mathbb{R}^N$ and $N \times N$ diagonal weighting matrix A with $a(1), \ldots, a(N)$ along its diagonal, $\|z\|_A = \sqrt{\sum_{q=1}^{N}(\sqrt{a(q)}z(q))^2} = \|\sqrt{A}z\|$. Using this, we can rewrite f_k as

$$f_k(w_k) = \frac{1}{\mathrm{tr}(D_k)}\|\sqrt{D_k}V - \sqrt{D_k}\Phi w_k\|^2,$$

which shows that for classifier k we are operating in the inner product space defined by $\|\cdot\|^2_{D_k} =< D_k\cdot, \cdot >$. Hence, we perform an approximation of V into the approximation space $\{\sqrt{D_k}\Phi w_k : w_k \in$

$\mathbb{R}^L\}$. As commonly known, minimising the above mean-squared error equals an orthogonal projection into the approximation space, which can be expressed by the linear transform

$$\tilde{V}_k = \Pi_k V,$$

where Π_k is the $N \times N$ projection matrix for classifier k, given by

$$\Pi_k = \Phi(\Phi' D_k \Phi)^{-1} \Phi' D_k.$$

This matrix performs a projection into the subspace spanned by the column vectors of $\sqrt{D_k}\Phi$. The $L \times L$ matrix $\Phi' D_k \Phi$ is invertible if the basis functions $\phi_1, \dots \phi_L$ are linearly independent, and if classifier k matches a sufficient number of states, i.e. $\mathrm{tr}(I_{S_k}) \geq L$.

Using the optimal approximation $\Pi_k V$ that minimises $\|V - \Phi w_k\|_{D_k}$, we can describe the minimal approximation error simply by

$$\frac{1}{\mathrm{tr}(D_k)} \|V - \Pi_k V\|^2_{D_k}.$$

In terms of vector spaces, this error is the weighted norm of the vector $V - \Pi_k V$, which is a vector that is orthogonal to the approximation space. Therefore, it fulfils the requirement of the Principle of Orthogonality (Theorem 3.2), that the approximation is orthogonal to the error. Even though the Principle of Orthogonality deals with approximation by sampling, the (in this case more appropriate) Theorem 3.1 is equivalent to the Principle of Orthogonality in its limit $t \to \infty$.

The matrix notation becomes particularly useful when investigating the interaction between LCSs function approximation and synchronous reinforcement learning, as done in [15].

4 Algorithmic Approaches

In this section we will discuss several algorithmic implementations to function approximation that are currently in use in LCS. Additionally, we will introduce a new method that is based on the Kalman filter and outperforms - at least in theory - all currently applied methods.

4.1 Steepest Gradient Descent

Steepest Gradient Descent is a well-known method for function minimisation, that performs small steps along the steepest gradient towards the next local minimum. Let us consider the mean-squared error f_k according to Eq. (3) as the function we want to minimise. Then, the steepest gradient descent algorithm is defined by

$$w_{k,t} = w_{k,t} - \alpha_t \frac{1}{2} \nabla_{w_k} f_k(w_{k,t}), \tag{7}$$

where $\alpha_t > 0$ is the scalar step size at time t, and $\nabla_{w_k} f_k(w_{k,t})$ is the gradient of f_k w.r.t. w_k, which is, according to Eq. (3),

$$\nabla_{w_k} f_k(w_{k,t}) = -2 \left(\mathbb{E}_k(V\phi) + \mathbb{E}_k(\phi\phi') w_{k,t} \right).$$

The algorithm starts at an arbitrary weight vector $w_{k,0}$ and reduces the mean-squared error f_k with each step along the gradient. As the error function is a convex function, moving along the steepest gradient will under some assumptions lead us to the only minimum and with it to the optimal approximation.

4.1.1 Stability Criteria

By definition, the step size α_t can change with time. If it is kept constant, that is $\alpha_t = \alpha$ for all $t \geq 0$, and the gradient $\nabla_{w_k} f_k(w_{k,t})$ is Lipschitz continuous[9], then the steepest gradient descent method is guaranteed to converge to the minimum of the function, if that minimum exists [6, Prop 3.4]. It is possible to show that the Lipschitz continuity holds if the Hessian $\nabla^2_{w_k} f_k(w_{k,t})$ is bounded over $\mathbb{R}^L$, which depends on the definition of the basis functions $\phi_1, \dots, \phi_L$.

Another condition for the stability of steepest gradient descent, which is easier to evaluate, is for α to hold

$$0 < \alpha < \frac{2}{\lambda_{k,max}}, \tag{8}$$

where $\lambda_{k,max}$ is the largest eigenvalue of the feature vector correlation matrix $\mathbb{E}_k(\phi\phi')$ [20, Ch. 4]. Hence, stability conditions are highly dependent on the choice of the basis functions.

[9] A function $f : M \to \mathbb{R}$ is Lipschitz continuous, if there exists a finite constant scalar K such that $\|f(a) - f(b)\| \leq K\|a - b\|$ for any $a, b \in M$. The magnitude K is a measure of the continuity of the function f.

4.1.2 Time Constant Bounds

Similarly, the rate of convergence is also dependent on the eigenvalues of the feature vector correlation matrix. Let τ be the *time constant*[10] of the weight vector update. Then this time constant is bounded by

$$\frac{-1}{\ln(1-\alpha\lambda_{k,max})} \leq \tau \leq \frac{-1}{\ln(1-\alpha\lambda_{k,min})}, \tag{9}$$

where $\lambda_{k,max}$ and $\lambda_{k,min}$ are the largest and the smallest eigenvalue of $\mathbb{E}_k(\phi\phi')$ respectively [20, Ch. 4]. Additionally, if the eigenvalues are widely spread, which indicates ill-conditioned features, then the settling time of the gradient descent algorithm is limited by the smallest eigenvalue $\lambda_{k,min}$ [6, Ch. 3].

4.1.3 Applicability

As given by the algorithm definition, the steepest gradient descent method requires knowledge of the gradient $\nabla_{w_k} f_k(w_{k,t})$ for each step that it takes. Hence, we would have to have full knowledge of the mean-squared error, which requires full knowledge of our function V at each step. We could approach this by approximating the function over a finite number of steps, but we could never be sure of the quality of that approximation. Hence, we will treat the algorithm as a theoretical tool rather than one that we will use for implementation in LCSs. It was discussed here, as it shares some theoretical properties with the Least Mean Squares that we will describe in the next section.

Example 4.1 (Stability Criteria and Time Constant for some Classifier Types). Let us start with investigating averaging classifiers, defined by a feature vector of $\phi(i) = (1)$ for all $i \in S$, giving one eigenvalue $\lambda = 1$ for $\mathbb{E}_k(\phi\phi') = \mathbb{E}(I_{S_k})^{-1}\mathbb{E}(I_{S_k})$. Hence, according to Eq. (8) the steepest gradient descent method is stable for $0 \leq \alpha \leq 2$. Equation (9) gives its time constant as $\tau = -\ln(1-\alpha)^{-1}$. This confirms that the approximation responds more rapidly with a higher step size, as we would intuitively expect.

We can apply the same analysis to classifiers that approximate first-order polynomials, as given by the feature vector $\phi(i) = (1, i)'$. That gives the feature vector correlation matrix

$$\mathbb{E}_k(\phi\phi') = \begin{pmatrix} 1 & i \\ i & i^2 \end{pmatrix}$$

with the eigenvalues $\lambda_1 = 0, \lambda_2 = 1 + \mathbb{E}_k(i^2)$. Hence, for steepest gradient descent to be stable, the step size has to obey

$$0 \leq \alpha \leq \frac{2}{1 + \mathbb{E}_k(i^2)},$$

which demonstrates that large state values allow for only small step sizes. The time constant is bounded by

$$\frac{-1}{\ln(1-\alpha(1+\mathbb{E}_k(i^2)))} \leq \tau \leq \infty,$$

which shows that a large eigenvalue spread pushes the time constant towards infinity, resulting in very slow convergence. Therefore, the convergence rate of steepest gradient descent depends frequently on the range of the features[11].

4.2 Least Mean Square Algorithm

The Least Mean Square (LMS) algorithm is very much related to steepest gradient descent, but rather than performing gradient descent on the full gradient of the function, it performs gradient descent on a current local approximation. For this reason it is also called the *Stochastic Incremental Steepest Gradient Descent* algorithm, or *ADALINE*, or, after their developers Widrow and Hoff [33], the *Widrow-Hoff Update*.

Let us consider the mean-squared error (Eq. (3)), or its step-wise approximation (Eq. (4)), both of which take

$$I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))^2$$

[10] The time constant is a measure of the *responsitivity* of a dynamic system. A low time constant means that the system responds quickly to a changing input. Hence, it is inversely proportional to the rate of convergence.

[11] A similar analysis was done in [25], but there the stability criteria for steepest gradient descent were applied to the LMS algorithm.

as the error for state i_t, which is our observed state at time t. The LMS applies the same update equation as steepest gradient descent (see Eq. (7)), but rather than taking the overall gradient, it approximates it by the gradient of the error at time t, as given above. That leads to the LMS update

$$w_{k,t+1} = w_{k,t} + \alpha_t I_{S_k}(i_t)\phi(i_t)\left(V(i_t) - w'_{k,t}\phi(i_t)\right). \tag{10}$$

As the gradient estimate is only based on the current state, this method suffers from *gradient noise*. Due to this noise, a constant step size α will cause the method to perform random motion close to the optimal approximation [20, Ch. 5].

4.2.1 Misadjustment due to Local Gradient Estimate

The difference between the mean-squared error $f_k(w_k)$ and the minimum estimation error of the LMS algorithm is called the *excess mean squared estimation error*. The ratio between the excess mean squared estimation error and the minimum mean-squared error is called the *misadjustment*, which is a measure of how far away the convergence area of LMS is from the optimal approximation. The estimation error value for some small constant step size α can, according to [20, Ch. 5], be estimated by

$$f_k(w_k) + \frac{\alpha f_k(w_k)}{2}\sum_{j=1}^{J}\lambda_j,$$

where $f_k(w_k)$ stands for the minimum mean-squared error, and λ_j is the jth out of J eigenvalues of the feature vector correlation matrix $\mathbb{E}_k(\phi\phi')$. This shows that the excess mean squared estimation error is i) always positive, and ii) is proportional to the step size α. Thus, reducing the step size will also reduce the misadjustment due to local gradient estimation. Indeed, under the assumption that $\sum_{t=0}^{\infty}\alpha_t = \infty$ and $\sum_{t=1}^{\infty}\alpha_t^2 < \infty$, the Lipschitz continuity of the gradient, and some Pseudogradient property of the gradient, we can guarantee convergence to the optimal approximation [6, Prop. 4.1].

4.2.2 Stability Criteria

As we are more interested in constant step sizes, we will give guidelines about how to set a constant step size to provide stability of the method. As described in [20, Ch. 6], necessary conditions for long *traversal filters*, i.e. filters that keep a high number of past observations in memory, can be given, if the maximal value of the power spectral density of the inputs is known.

In our case, the filter is of length one, as it only uses the current observation for its approximation. In that case, no concrete bounds on the step size can currently be given. However, if the step size is small if compared with λ_{max}^{-1}, where λ_{max} is the largest eigenvalue of the feature vector correlation matrix, then the small-step theory can be applied which leads to the bounds

$$0 < \alpha < \frac{2}{\lambda_{max}}.$$

Those bounds are equal to the stability criteria for the steepest gradient descent method, as given by Eq. (8).

4.2.3 Average Time Constant

Let us define the average eigenvalue at time t by

$$\lambda_{av,t} = \frac{1}{J}\sum_{j=1}^{J}\lambda_{j,t}$$

where $\{\lambda_{1,t}, \lambda_{J,t}\}$ is the spectrum of the local feature vector matrix $\phi(i_t)\phi(i_t)'$. Then, according to [20, Ch. 5], the average time constant can be estimated by

$$\tau_{av,t} \approx \frac{1}{2\alpha\lambda_{av,t}}.$$

The use of this time constant is questionable, as it depends on the current state and is therefore not actually a constant. However, we can get an approximation of a global time constant by taking the expectation of the above local estimate. This results in a similar time constant as for the steepest gradient descent algorithm (see Eq. (9)). Overall, similar to steepest gradient descent, the rate of

convergence is still governed by the spread of the eigenvalues of the feature vector correlation matrix $\mathbb{E}_k(\phi\phi')$.

As discussed before, the misadjustment is proportional to the step size. On the other hand, the time constant is inversely proportional to the step size. Hence, we have conflicting requirements and can either aim for a low estimation error or a fast rate of convergence, but we will not be able to satisfy both requirements with anything other than a compromise.

4.2.4 Normalised LMS Algorithm

As already pointed out in [37], and can be seen from Eq. (10), the magnitude of the weight update is directly proportional to the feature vector $\phi(i_t)$, causing *gradient noise amplification* [20, Ch. 6]. This problem can be overcome by weighting the correction by the squared Euclidean norm of the feature vector. Hence, the update equation changes to

$$w_{k,t+1} = w_{k,t} + \alpha_t I_{S_k}(i_t) \frac{\phi(i_t)}{\|\phi(i_t)\|^2} \left(V(i_t) - w'_{k,t}\phi(i_t)\right).$$

This update equation can also be derived by calculating the weight update vector that minimises the norm of the weight change $\|w_{k,t+1} - w_{k,t}\|^2$, subject to the constraint

$$I_{S_k}(i_t) w'_{k,t+1}\phi(i_t) = V(i_t).$$

As such, the normalised LMS filter can be seen as a solution to a constrained optimisation problem.

Regarding stability, the step size parameter α is now weighted by the inverted squared norm of the feature vector. Hence, stability in the mean-squared error sense is dependent on the current state. The lower bound is still 0, and the upper bound will be generally larger than 2 if the state values are overestimated, and smaller than 2 otherwise. The optimal step size, located at the largest value of the mean-square deviation, is in the centre of the two bounds [20, Ch. 6].

As expected, the normalised LMS algorithm features a rate of convergence that is higher than that for the standard LMS filter, as demonstrated in simulation [14]. One drawback of this modification is that $\|\phi(i_t)\|^2$ has to be checked for being zero, to avoid divisions by zero. In that case, no weight update needs to be performed, as $\|\phi(i_t)\|^2 = 0$ implies that $\phi(i_t)$ is zero in all its elements.

4.3 The Kalman Filter and Recursive Least Squares Algorithm

The Kalman filter is a recursive solution to the discrete-data linear filtering problem [21]. In this section we will apply the Kalman filter to the approximation task and simultaneously to tracking the approximation error. Being able to use the Kalman filter for that task is of advantage as "[...] the Kalman filter is optimal with respect to virtually any criterion that makes sense" [26, Ch. 1].

4.3.1 The System Model

Let us define the linear Kalman-Bucy system model [22] by the two system equations

$$\begin{aligned} x_{t+1} &= F_t x_t + G_t w_t, \\ z_t &= H'_t x_t + v_t, \end{aligned}$$

where x_t is the system state vector, F_t is the process matrix, w_t is the process noise with its transformation matrix G_t, z_t is the measurement, H_t is the measurement matrix, and v_t is the measurement noise, all at time t. Hence, the model describes how a process modifies a system state $\{x_0, x_1, \dots\}$ over time, and how this affects the observations $\{z_0, z_1, \dots\}$ of the system. Note that the notation used in above system model is not to be confused with the notation of the rest of the paper, but will be put into relation soon.

The Kalman filter applies a Gaussian model of the system state and the measurement, and assumes that the process and measurement noise to be white and Gaussian. With these assumptions, it tracks the optimal system state estimate, given all available measurements and knowledge of the noise covariance matrix.

To apply the Kalman filter to our approximation task, we will assume that the system can be described by a linear architecture as given by Eq. (2). As the function V we want to approximate is time-invariant, and not perturbed by noise, the whole system is assumed to be time-invariant, which leads to the process matrix being the identity matrix, and the process noise being zero. An observation of the system state, however, is determined by the feature vector of the current state i_t and by the

measurement noise, which captures the deviation of the real function V from the linear architecture and our choice of features. The aim of the Kalman filter is to update the Gaussian model of the system state with every observation, to get the best correspondence between the system state estimate and all past observations. As we will see later, this aim is equivalent to minimising the approximated mean-squared error as given by Eq. (4).

Let us denote the time-invariant system state w.r.t. classifier k by W_k. As the system state is time-invariant and the process noise is zero, we only need to consider the measurement part of the Kalman-Bucy model, which can be rewritten as

$$V_t = W_k' \phi(i_t) + \xi_{k,t}, \tag{11}$$

where V_t is the observation at time t, with measurement noise $\xi_{k,t} \in \mathbb{R}$. The measurement noise is assumed to be of zero mean, white, Gaussian, and independent of V_t and W_k. Its variance is known and given by

$$\mathbb{E}(\xi_{k,t_1}\xi_{k,t_2}) = \begin{cases} \Xi_{k,t_1} & \text{if } t_1 = t_2, \\ 0 & \text{otherwise,} \end{cases}$$

where $\Xi_{k,t}$ denotes the measurement noise variance concerning classifier k at time t. In other words, when measuring value V_t, the measurement noise is modelled by a Gaussian $N(0, \Xi_{k,t})$.

4.3.2 Bayesian Estimation Update as the Kalman Filter Algorithm

Let $W_{k,t}$ be the estimate of the system state W_k at time t. From the system model (Eq. (11)) it is clear that this estimate is Gaussian, and that V_t and $W_{k,t}$ are jointly Gaussian. Hence, the estimate is fully defined by $W_{k,t} = N(w_{k,t}, \Sigma^w_{k,t})$, where $w_{k,t} \in \mathbb{R}^L$ is the mean and $\Sigma^w_{k,t} \in \mathbb{R}^L \times \mathbb{R}^L$ is the $L \times L$ covariance matrix. A more detailed description and analysis of the distribution and interrelation of the different system variables can be found in [26, Ch. 5] or [1, Ch. 1].

As discussed before, we want to update our estimate with every additional observation of a matched state, at time t given by the function value $V(i_t)$, the measurement noise variance $\Xi_{k,t}$, and the feature vector $\phi(i_t)$. Even though we will describe in Section 4.3.5 how to estimate the measurement noise variance together with the state estimate, let us for now assume that $\Xi_{k,t}$ is known. The Bayesian update of the system state estimate is performed by conditioning it on the latest observation $V_t = N(V(i_t), \Xi_{k,t})$, that is (according to [1, Ch. 3]):

$$\begin{aligned} \mathbb{E}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t})) &= \mathbb{E}(W_{k,t-1}) + \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}(V(i_t) - \mathbb{E}(V_t)), \\ \mathrm{cov}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t})) &= \mathrm{cov}(W_{k,t-1}) - \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}\mathrm{cov}(V_t, W_{k,t-1}). \end{aligned}$$

For simplicity, we write $\mathrm{cov}(X)$ for $\mathrm{cov}(X, X)$. This Bayesian update acts as the core of the Kalman filter [26, Ch. 5].

The old estimate $W_{k,t-1} = N(w_{k,t-1}, \Sigma^w_{k,t-1})$ conditioned on the observation V_t gives the new estimate

$$W_{k,t} = N(w_{k,t}, \Sigma^w_{k,t}) = N\left(\mathbb{E}(W_t|V_t = N(V(i_t), \Xi_{k,t})), \mathrm{cov}(W_{k,t-1}|V_t = N(W(i_t), \Xi_{k,t}))\right).$$

Together with the evaluated expectations, covariances and variances, and restricting the update to matched states (see Appendix A.1 for derivations) that gives the Kalman filter algorithm:

$$\begin{aligned} \zeta_{k,t} &= I_{S_k}(i_t)\Sigma^w_{k,t-1}\phi(i_t)\left(I_{S_k}(i_t)\phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t}\right)^{-1}, & (12) \\ w_{k,t} &= w_{k,t-1} + \zeta_{k,t}(V(i_t) - w'_{k,t-1}\phi(i_t)), & (13) \\ \Sigma^w_{k,t} &= \Sigma^w_{k,t-1} - \zeta_{k,t}\phi(i_t)'\Sigma^w_{k,t-1}. & (14) \end{aligned}$$

This form of the Kalman filter update is commonly called the *Covariance Form*. The measurement residual $V(i_t) - w'_{k,t-1}\phi(i_t)$ is the difference between the observation $V(i_t)$ and its estimate $w'_{k,t-1}\phi(i_t)$ before $V(i_t)$ is known. The *Kalman gain* ζ_t determines how much the current estimate is corrected.

From the update equation we can see that as the measurement noise variance $\Xi_{k,t}$ approaches zero, the gain $\zeta_{k,t}$ weights the residual more heavily. On the other hand, as the weight covariance $\Sigma^w_{k,t}$ approaches zero, the gain $\zeta_{k,t}$ assigns less weight to the residual [32]. This is the behaviour that we would intuitively expect, as low-noise measurements should be valued higher than high-noise measurements.

4.3.3 Inverse Covariance Form

Using the Kalman filter for estimating the system state requires setting initial values for the estimate $W_{k,-1}$. In many cases, we do not have any knowledge about what the correct values might be and setting random initial values will cause an unnecessary bias. Complete lack of information about the initial system state can be modelled as the limiting case of certain eigenvalues of $\Sigma^w_{k,-1}$ going to infinity [26, Ch. 5.7]. That will work in theory, but would cause problems in implementation due to large numerical errors when evaluating the Kalman gain according to Eq. (12).

Another approach to the Kalman update is to operate on the inverse of the covariance rather than the covariance itself (for derivation see Appendix A.2), resulting in the update

$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)', \tag{15}$$

$$(\Sigma^w_{k,t})^{-1} w_{k,t} = (\Sigma^w_{k,t-1})^{-1} w_{k,t-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t} V(i_t). \tag{16}$$

Note that rather updating the weight vector $w_{k,t}$ directly, we are now performing updates on the vector $(\Sigma^w_{k,t})^{-1} w_{k,t}$. The weight estimate can be recovered by

$$w_{k,t} = \left[(\Sigma^w_{k,t})^{-1}\right]^{-1} \left[(\Sigma^w_{k,t})^{-1} w_{k,t}\right].$$

That shows that we are only required to perform a matrix inversion for computing the value of the weight vector, but not for the update itself. This is of advantage if our initial inverse covariance $(\Sigma^w_{k,-1})^{-1}$ is singular, which is the case if we set it to zero to avoid initial estimation bias. Hence, we can perform updates until the inverse covariance attains full rank, which subsequently allows for a unique weight estimate. This is not the case for the Kalman filter in its Covariance form, as given by Eq. (12), (13) and (14).

Example 4.2 (Inverse Covariance Form for Singular Inverse Covariance). Let us consider a classifier with feature vectors $\phi(i) = (1, i)'$. Setting $(\Sigma^w_{k,-1})^{-1} = 0$ allows us to avoid any initial bias. Let us now assume that at time $t = 0$ we observe value $V(i_0)$ in state i_0, which is matched by classifier k. According to Eq. (15) and (16), the classifier parameters of classifier k are updated to

$$(\Sigma^w_{k,0})^{-1} = \begin{pmatrix} \Xi^{-1}_{k,0} & \Xi^{-1}_{k,0} i_0 \\ \Xi^{-1}_{k,0} i_0 & \Xi^{-1}_{k,0} i_0^2 \end{pmatrix},$$

$$(\Sigma^w_{k,0})^{-1} w_{k,0} = \begin{pmatrix} \Xi^{-1}_{k,0} V(i_0) \\ \Xi^{-1}_{k,0} i_0 V(i_0) \end{pmatrix}.$$

Clearly, $w_{k,0}$ cannot be computed due to the singularity of $(\Sigma^w_{k,0})^{-1}$. However, we can still perform further updates of the inverse covariance until it is of full column rank and can be inverted. Until then, we can use the pseudo-inverse to obtain first estimates of the weight vector, resulting in

$$w_{k,0} = \frac{1}{1 + i_0^2} \begin{pmatrix} V(i_0) \\ i_0 V(i_0) \end{pmatrix}.$$

4.3.4 Equivalence to Recursive Least Squares

In this section we will demonstrate that our application of the Kalman filter is equivalent to the Recursive Least Squares (RLS) algorithm, applied to a measurement noise weighted error function. Due to its lengthly deviation we will only state the result, of which the proof can be found in Appendix A.3:

Theorem 4.1 (Recursive Least Squares Algorithm). *Given the sequence of states $\{i_0, i_1, \dots\}$, the sequence of features $\{\phi(i_0), \phi(i_1), \dots\}$, the sequence of function values $\{V(i_0), V(i_1), \dots\}$, and the sequence of measurement noise variances $\{\Xi_{k,0}, \Xi_{k,1}, \dots\}$, we aim at minimising the error*

$$\sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}(V(i_m) - w'_{k,t}\phi(i_m))^2$$

at time t.

1. *Starting with $(\Sigma^w_{k,-1})^{-1} = 0$, the iterative update*

$$w_{k,t} = w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}\left((\Sigma^w_{k,t1})^{-1}\right)^{-1}\phi(i_t)\left(V(i_t) - w'_{k,t-1}\phi(i_t)\right),$$

$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)',$$

gives the sequence of weight vectors $w_{k,t}$ that minimise above error.

2. *Given that the measurement noise variance is time-invariant, that is* $\Xi_{k,t} = \Xi_k$ *for all* $t = 0, 1, \ldots$, *then*

   - *the measurement noise variance* $\Xi_{k,t}$ *can be omitted from the above weight and covariance update equations,*
   - *the tracked weight vector conforms to the Principle of Orthogonality (Theorem 3.2), and*
   - *the iterative update*

   $$(c_{k,t} - 1)\varepsilon_{k,t} = (c_{k,t-1} - 1)\varepsilon_{k,t-1} + I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))(V(i_t) - w'_{k,t-1}\phi(i_t)),$$

   starting with $\varepsilon_{k,-1} = 0$ *tracks the approximated mean-squared error (see Eq. (4)).*

It is apparent that the RLS inverse covariance matrix update is equivalent to Eq. (15). Showing equivalence to the weight update in Inverse Covariance Form (Eq. (16)) is achieved by pre-multiplying the RLS weight update by $(\Sigma^w_{k,t})^{-1}$:

$$\begin{aligned}(\Sigma^w_{k,t})^{-1} w_{k,t} &= (\Sigma^w_{k,t})^{-1} w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)V(i_t) - I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)w'_{k,t-1}\phi(i_t) \\ &= (\Sigma^w_{k,t-1})^{-1} w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t).\end{aligned}$$

Hence, the Kalman filter tracks the optimal approximation given a noise variance-weighted error function, and is therefore optimal in the mean-squared error sense.

4.3.5 Minimum Model Error Approximation

The Kalman system model (Eq. (11)) attributes any deviation of the observations of V from the chosen linear architecture to the measurement noise $\xi_{k,t}$. As this measurement noise gives us information about how well the linear architecture approximates the observed function values, we want to estimate the noise variance simultaneously to approximating the function. For that purpose we will assume that the measurement noise variance is time-invariant, i.e. $\Xi_{k,t} = \Xi_k$ for all $t = 0, 1, \ldots$, and will adopt the *Minimum Model Error (MME)* philosophy, that aims at finding the approximation that minimises the measurement noise variance. The MME estimate was developed in [27] to estimate the system parameters and system model error while minimising the latter. It is based on the *Covariance Constraint* condition, which states that the measurement-minus-estimate error covariance must match the measurement-minus-truth error covariance, that is

$$I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))^2 = \Xi_k.$$

Given that constraint, and the assumption of not having any process noise, we can define the model error at time t by weighting the left-hand side of above equation by the inverted right-hand side, that is

$$\frac{1}{1 - c_{k,t}}\Xi_k^{-1} \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{w,t}\phi(i_m))^2.$$

Note that this is equivalent to the approximated mean-squared error $\varepsilon_{k,t}$ (Eq. (4)) scaled by the inverted noise variance Ξ_k^{-1}. As the noise variance can be expressed by the mean-squared error $f_k(w_k)$ (Eq. (3)), we can interpret above approximation cost as the approximated mean-squared error in relation to the real mean-squared error, which will converge to 1 with $t \to \infty$. Additionally, it shows that we can use the approximated mean-squared error $\varepsilon_{k,t}$ as an appropriate approximation for the measurement noise variance Ξ_k.

Let us investigate the effect of assuming a time-invariant measurement noise variance. As already discussed, this causes the minimisation goal to be independent of the measurement noise variance. Therefore, when tracking the weight vector and covariance matrix, we can set it to any value without changing the outcome, for any form of the Kalman update. Additionally, we can approximate the measurement noise variance by tracking the approximated mean squared error according to Theorem 4.1. Overall, as we know that for a finite S the approximated mean squared error converges to the mean squared error (see Theorem (3.5)), and as we are tracking the optimal approximation that minimises the approximated mean squared error, we can guarantee convergence of this approximation to the optimal approximation as given by Theorem 3.1, with probability 1.

Example 4.3 (Kalman Filter Update for Averaging Classifiers). Let us consider an averaging classifier k, using feature vector $\phi(i) = (1)$ for all $i \in S$. As we want to estimate the measurement

noise variance simultaneously to finding the optimal approximation, we will assume it to be a constant Ξ_k. To avoid any initial bias, let $(\Sigma^w_{k,0})^{-1} = 0$. Then according to Eq. (15), the inverse covariance is updated by

$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\Xi_k^{-1} = \Xi_k^{-1} c_{k,t},$$

which is the match count, scaled by the inverse measurement noise variance. Hence, the covariance (which is in this case a variance) of our weight estimate decreases exponentially with the number of matched states that were visited. Hence, our estimate will become increasingly more accurate.

The weight update is performed according to Eq. (16), that is

$$(\Sigma^w_{k,t})^{-1} w_{k,t} = (\Sigma^w_{k,t-1})^{-1} w_{k,t-1} + I_{S_k}(i_t)\Xi_k^{-1} V(i_t) = \Xi_k^{-1} \sum_{m=0}^{t} I_{S_k}(i_m) V(i_m),$$

giving for $w_{k,t}$

$$w_{k,t} = \frac{(\Sigma^w_{k,t})^{-1} w_{k,t}}{(\Sigma^w_{k,t})^{-1}} = \frac{1}{c_{k,t}} \sum_{m=0}^{t} I_{S_k}(i_m) V(i_m),$$

which is the average over the function values of the state sequence $\{i_0, \ldots, i_1\}$, when only considering matched states. Note that due to the constant measurement noise variance, this weight estimate is independent of the measurement noise variance.

As the weight vector is optimal for each t, applying the error approximation update from Theorem 4.1 tracks the function sample variance (see Example 3.1). Note that the variance of the weight estimate indicates how sure we are about the current estimate and is a different measure than the sample variance, which indicates how well the linear architecture is able to approximate the function of its matching states.

Interestingly, XCS applies MAM update that is equivalent to averaging the input for the first α^{-1} inputs, where α is the step size, and then tracking the input using the LMS algorithm [35]. In other words, it bootstraps its weight estimate using the Kalman filter algorithm with a measurement noise variance of 1, and then continues tracking of the input using the LMS algorithm. Note that this is only the case for XCS applying averaging classifiers, and does not apply to XCS-derivates that apply more complex function approximation, such as XCSF [37]. Even though it is not explicitly stated in [37], we assume that the MAM update was not used in those XCS-derivates.

4.3.6 Yet Another Possible Implementation

In addition to the standard form of the Kalman filter, as given by the Covariance form, the Inverse Covariance Form, and the RLS algorithm, we present a computationally cheap implementation which we have used for experiments, but for which we cannot give any guarantees on its numerical stability when applied for long time frames.

As our developments aim at application in LCSs, we want to approximate the error while tracking the optimal approximation. That calls for using a constant measurement noise variance, which consequently allows us to use the approximated mean-squared error (Eq. (4)) as our approximation goal. Hence, we can utilise the Principle of Orthogonality (Theorem 3.2), of which we can rewrite the condition for the optimal weight vector by

$$\left(\sum_{m=0}^{t} I_{S_k}(i_m) \phi(i_m) \phi(i_m)' \right) w_k = \sum_{m=0}^{t} I_{S_k}(i_m) V(i_m) \phi(i_m),$$

Let us denote the matrix in brackets of the left-hand side by $A_{k,t}$ and the vector of the right-hand side by $b_{k,t}$. Having those, we can compute the weight vector $w_{k,t}$ by

$$w_{k,t} = A_{k,t}^{-1} b_{k,t},$$

given that $A_{k,t}$ is non-singular. Vector $b_{k,t}$, starting with $b_{k,-1} = 0$, can be updated by

$$b_{k,t} = b_{k,t-1} + I_{S_k}(i_t) V(i_t) \phi(i_t).$$

To avoid taking the inverse every time when we are updating $A_{k,t}$, we can apply the Sherman-Morisson formula to operate on the inverse, that is

$$A_{k,t}^{-1} = A_{k,t-1}^{-1} - I_{S_k}(i_t) \frac{A_{k,t-1}^{-1} \phi(i_t) \phi(i_t)' A_{k,t-1}^{-1}}{1 + I_{S_k}(i_t) \phi(i_t)' A_{k,t-1}^{-1} \phi(i_t)}.$$

The approximation error is again updated according to Theorem 4.1.

To recover the covariance matrix $\Sigma^w_{k,t}$, we can give its inverse by

$$(\Sigma^w_{k,t})^{-1} = \Xi_k^{-1} \sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)',$$

which is matrix $A_{k,t}$ scaled by the measurement noise variance Ξ_k. As the error $\varepsilon_{k,t}$ is the current best approximation of the measurement noise variance, we can calculate the covariance matrix by

$$\Sigma^w_{k,t} = \varepsilon_{k,t} A^{-1}_{k,t}.$$

The disadvantage of this update is that we require to set matrix $A_{k,t}$ to an initial value $A_{k,-1}$, which introduces a bias to the approximation. This bias can be kept low by setting $A_{k,-1} = I\delta$, where δ is a large scalar and causes the initial inverse $A^{-1}_{k,t}$ to be small. This can still introduce numerical instabilities, but should be sufficient for most applications. Additionally, the update still converges to the same optimal approximation as $t \to \infty$. Setting $A^{-1}_{k,-1} = 0$ is invalid as it will cause $A^{-1}_{k,t} = 0$ for all $t = 0, 1, \ldots$.

4.3.7 Stability of the Kalman Filter

The Covariance Form of the Kalman filter, as summarised by Eq. (12), (13) and (14) suffers from serious numerical difficulties, as described in [20, Ch. 6]. Considering Eq. (12) and (14), for example, the covariance matrix is update by the difference between two non-negative matrices. Unless the system is numerically completely accurate, the result of the subtraction may not be non-negative definite, which is not acceptable for a covariance matrix. Additionally, the property of symmetry of the covariance matrix might not be preserved due to numerical inaccuracy. The Inverse Covariance Form, given by Eq. (15) and (16), shows better properties but is still affected by the possible loss of symmetry.

A method of overcoming these stability issues is to use numerically stable unitary transforms at every iteration, as, for example, the *Square-Root Form* of the Kalman filter. This form is mathematically equivalent to the other Kalman filter forms, but only operates on the lower triangle of the covariance matrix and hence preserves symmetry. In addition, in that form the matrix is much less likely to become indefinite. We will not discuss any further details here, but refer the interested reader to [20, Ch. 11], [1, Ch. 6.5] or [26, Ch. 7].

4.3.8 Comparison to the LMS Filter

As stated in [20, Ch. 9], the RLS algorithm typically features an order of magnitude faster rate of convergence than that of the LMS algorithm. Additionally, its rate of convergence is less sensitive to the spread of eigenvalues of the feature vectors. As our use of the Kalman filter is equivalent to a form of the RLS algorithm, these findings also apply to the Kalman filter.

On the downside, the Kalman filter is both computationally and spatially more complex than the LMS filter. It requires keeping the covariance of the weight estimates, and employs matrix multiplication and eventual inversion, depending on the update type that is used. A detailed summary of the different forms of Kalman filters and their computational complexities can be found in [26, Ch. 7.8].

In the case of low computational power and high spatial constraints we recommend applying the LMS filter. However, in all other cases, the Kalman filter should be applied.

4.4 Experimental Confirmation

In this section we demonstrate the superiority of the Kalman filter when compared to the LMS algorithm. Additionally, we will discuss the relation of the presented algorithms to currently common implementations in LCS.

The two cases that we will deal with are i) averaging classifiers that have to approximate a noisy constant, and ii) classifiers that approximate parts of a sinusoid by a first-oder polynomial. We have chosen the first case to investigate the accuracy of the error approximation. The second case tests both the approximation of the weight vector and the mean squared error.

We will compare three kinds of classifiers:

Original XCS Classifiers [35] These classifiers apply the LMS algorithm for the averaging case (i.e. using a feature vector of $\phi(i) = (1)$ for all $i \in S$), and the normalised LMS algorithm for any other case (as introduced in [37]). In the averaging case, they will additionally perform MAM update,

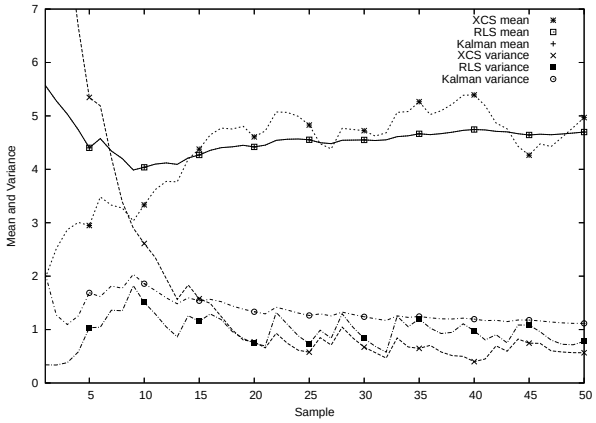


Figure 1: Comparing the approximation of the mean and the variance of values sampled from $N(5, 1)$.

as described in Example 4.3. The approximation error is tracked by applying the LMS algorithm to the current error of the error approximation $I_{S_k}(i_t)\left(I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))^2 - \varepsilon_{k,t-1}\right)^2$, which gives

$$\varepsilon_{k,t} = \varepsilon_{k,t-1} + \alpha_t I_{S_k}(i_t)\left((V(i_t) - w'_{k,t}\phi(i_t))^2 - \varepsilon_{k,t-1}\right).$$

To be precise, the error that is tracked in XCS is for no apparent reason the mean-absolute error $I_{S_k}(i_t)|V(i_t) - w'_{k,t}\phi(i_t)|$, but to be consistent with our error definition and to keep the results comparable, we have chosen to modify the algorithm to track the mean-squared error. For all experiments, the step size α is kept constant and set to the commonly applied value $\alpha = 0.2$. All other values are initialised to 0.

RLS Classifiers [24] The RLS Classifiers use the Recursive Least Squares algorithm, as introduced in Theorem 4.1, for its weight update, with a constant measurement noise variance of $\Xi_k = 1$[12]. Its covariance matrix $\Sigma^w_{k,t}$ is initialised to $\Sigma^w_{k,-1} = \delta I$ with $\delta = 1000$. Error update is performed in the same way as for the original XCS classifiers by the LMS algorithm. The step size is again set to $\alpha = 0.2$.

Kalman filter Classifiers Those classifiers apply the Kalman filter update under the Minimum Model Error philosophy, as described in this paper. The form of update used is described in Section 4.3.6. It is initialised with $\delta = 1000$.

4.4.1 Averaging Classifiers

In our first experiment we apply averaging classifiers, using a feature vector of $\phi(i) = (1)$, for all $i \in S$. Each classifier matches all states which is why we can reduce the system to a single state without the loss of generality. The sampled function values of that state are sampled from $N(5, 1)$, that is a normal distribution with mean 5 and variance 1. We have not considered noisy sampling in developing conditions for optimality and discussing the algorithms. However, by assuming that each sample describes the function value of a different state, the developed theory is still valid. The optimal values are $w_k(1) = 5$ and $f_k(w_k) = 1$.

Figure 1 shows the results over the first 50 samples. As the weight vector for the XCS classifier is initialised to $w_{k,-1} = 0$, it requires about 15 observations to get close to the optimal weight value. Due to considering only the local gradient, it then performs stochastic motion around that optimal point. The RLS and Kalman classifiers both apply a mathematically equivalent algorithm and therefore also have equivalent weight approximations. These approximations are obviously significantly more stable and less noisy, which is as expected, as they track the optimal approximation given all past observations.

The approximated error, given by the variance, reflects the same picture. Both XCS and RLS classifier use the LMS algorithm to update their error approximation and therefore require more initial time to get close to the optimal, and then perform stochastic motion around it. The XCS variance is particularly bad in the beginning, as its approximation of the mean is worse than for

[12]The standard RLS algorithm does not consider measurement noise, and therefore always operates under the condition of a constant measurement noise variance.

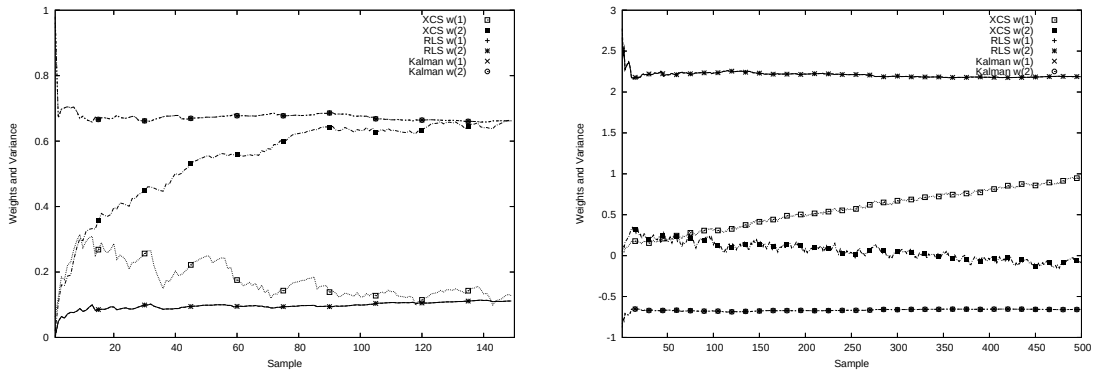


Figure 2: Approximating a sinusoid using a first-order polynomial. Graph (a) shows the weight values for the approximation over the range $[0, \frac{\pi}{2})$. Graph (b) shows the same over the range $[\frac{\pi}{2}, \pi)$.

the RLS classifier. The Kalman filter classifier again features better initial performance and higher stability in its approximation. Even though the RLS classifier seems to reside closer to the optimal error approximation, this is due to the initial bias of setting $e_{k,-1} = 0$. The Kalman filter classifier initially overestimates the variance, but its estimate is theoretically optimal and its initial deviation from the long-run optimal stems from the randomness of the sampling process.

4.4.2 Approximating a Sinusoid

The second experiment requires the classifiers to approximate a sinusoid $V(i) = \sin(i)$, using a first-oder polynomial, given by the feature vector $\phi(i) = (1, i)'$, for all $i \in S$. We employ two classifiers, 1 and 2, of which the first matches the range $S_1 = [0, \frac{\pi}{2})$, and the second matches the range $S_2 = [\frac{\pi}{2}, \pi)$. Evaluating the optimality conditions of Theorem 3.1 (see Appendix A.4 for derivation), we get the optimal weight vectors $w_1 \approx (0.115, 0.665)'$ and $w_2 \approx (2.202, -0.665)$, and equal mean-squared errors $f_1(w_1) = f_2(w_2) = 0.00394$. The approximation is evaluated in two runs, where in the kth run we uniformly sample states from S_k and only observe classifier k.

Figure 2 shows the first 150 observations for classifier 1 in (a), and the first 500 observations for classifier 2 in (b). As in the first experiment, the weight vectors are congruent for RLS and Kalman filter classifiers, as their approximation methods are mathematically equivalent. For both classifiers, the Kalman filter approximations feature better initial performance and higher stability if compared to XCS using the normalised LMS algorithm. What becomes particularly apparent in Figure 2(b) is that ill-conditioned features reduce the rate of convergence of the LMS algorithm, which was already postulated in Example 4.1 and is now confirmed by the slow convergence of XCS. As the error approximation behaves similar to the first experiment, with the Kalman filter demonstrating better initial values and higher stability than both the RLS and XCS classifiers (and even worse initial performance of XCS than in the first experiment), we have omitted the graph showing that error approximation.

5 Mixing Classifier Estimates

So far, we have concentrated on finding optimal approximations for single classifiers. This section concentrates on how the approximation of single classifiers can be assembled to give an approximation over the whole domain of the function V.

We have already described how mixing is performed in our framework (see Eq. (5)), and presented arguments why we should prefer averaging over approximations rather that aggregating them. The open question that we will consider in this section is how to set the mixing weights $\psi_k(i)$. For that purpose, we will assume a fixed set of classifiers that have all converged to the optimal approximation. Hence, we will drop the subscript $\cdot_t$, making the parameters time-independent, including $\varepsilon_{k,t}$, which we will simply denote by ε_k. Nevertheless, all findings from this section are also valid for the time-dependent case, as we can simply observe the LCS at a fixed time t and make the same investigations.

5.1 Accuracy-based Mixing

By definition, the mean-squared error of a classifier gives the average approximation error over its match set. Hence, a lower mean-squared error also implicates an on average higher approximation accuracy. As we aim at mixing the classifiers in a way to increase the overall approximation accuracy, making the mixing weight of a classifier inversely proportional to its approximation error seems intuitively correct.

In XCS [35], the first accuracy-based LCS, the accuracy of classifier k is separately calculated by some inverse power-function of the approximation error, and then additionally scaled relative to the accuracy of classifiers that match some states in S_k. In our opinion, such additional complexity is unnecessary, as scaling the accuracy w.r.t. other classifiers does not change its relative value w.r.t. those classifier, which is the value that is then considered for mixing. Note that in XCS, there is no explicit separation between the use of accuracy for mixing classifiers, and as a measure of a classifier's fitness for application in the evolutionary component of XCS. In our opinion, those two values do not necessarily have to be equal, and currently we only concentrate on the mixing weights, which we will define closely related to YCS [9] (a simplified version of XCS), that is

$$\psi_k(i) = \frac{I_{S_k}(i)\varepsilon_k^{-\nu}}{\sum_{p=1}^K I_{S_p}(i)\varepsilon_p^{-\nu}}, \tag{17}$$

where $\nu \in \mathbb{R}$ is a positive constant that allows additional control on the mixing weights. It is easy to check that a mixing weight defined like that satisfies the constraints given by our framework, that is for any $i \in S$, $\sum_{k=1}^K \psi_k(i) = 1$, and $\psi_k(i) = 0$ if $i \notin S_k$. Note that above equations is undefined for states for which no classifier matches, but in such cases it is apparent that we are unable to give an approximation.

5.2 Using the Maximum Likelihood Estimate

In deriving the update equations for the Kalman filter, we have modelled the weight estimate by a Gaussian, given by $N(w_k, \Sigma_k^w)$. The same thing can be said for the observation of V for state i, which can be modelled by a Gaussian with mean $\tilde{V}_k(i) = w_k'\phi(i)$ and the estimated variance of $\tilde{V}_k$, as given by $\phi(i)'\Sigma_k^w\phi(i) + \Xi_k$ (see Appendix A.1 for derivation). By having a model of the observation for each classifier, we can derive a mixed model of all classifiers by using the Maximum Likelihood Estimate (MLE).

The MLE defines the value of highest likelihood by the maximum of the *likelihood function*, which is the product of the probability density function of the models of all classifiers. As deriving the logarithm of the likelihood function is usually easier and gives the same maximum, we will use this *log-likelihood*, denoted by Λ, to derive the maximum. Let us fix a state $i \in S$ and define $\mu_k = w_k'\phi(i)$ as the mean and $\sigma_k = \sqrt{\phi(i)'\Sigma_k^w\phi(i) + \Xi_k}$ as the standard deviation of the Gaussian model for classifier k. Then, substituting for the probability density function of a Gaussian, the log-likelihood Λ for our approximation $\tilde{V}(i)$ is given by

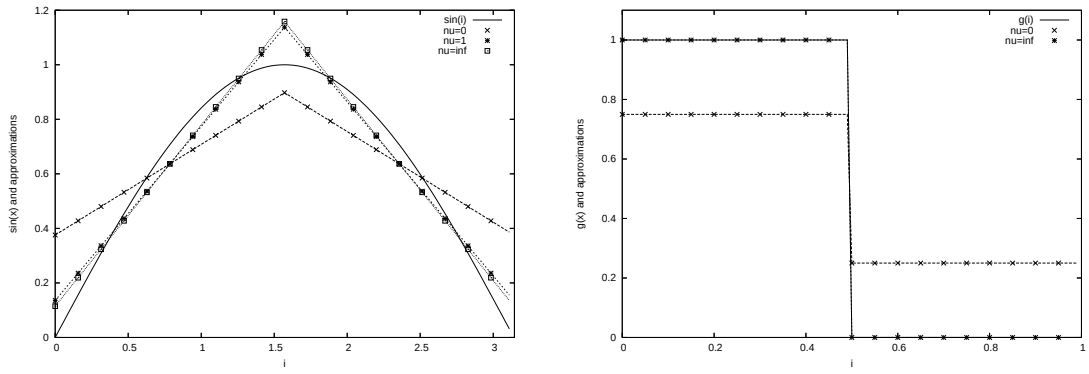
$$\Lambda(\tilde{V}(i)) = \sum_{k=1}^K I_{S_k}(i) \ln\left(\frac{1}{\sqrt{(2\pi)}\sigma_k} \mathrm{e}^{\frac{(\tilde{V}(i)-\mu_k)^2}{2\sigma_k^2}}\right) = \sum_{k=1}^K I_{S_k}(i) \ln\left(\frac{1}{\sqrt{2\pi}\sigma_k}\right) - \frac{1}{2}\sum_{k=1}^K I_{S_k}(i)\frac{(\tilde{V}(i)-\mu_k)^2}{\sigma_k^2}.$$

Adding $I_{S_k}(i)$ is equivalent to taking the probability density function of classifier k to the power of $I_{S_k}(i)$, and ensures that we ignore classifiers that do not match state i. We can get the unique maximum of the convex log-likelihood function by setting its first derivative w.r.t. $\tilde{V}(i)$ to zero, which results in

$$\tilde{V}(i) = \sum_{k=1}^K \mu_k \frac{I_{S_k}(i)\sigma_k^{-2}}{\sum_{p=1}^K I_{S_p}(i)\sigma_p^{-2}},$$

giving the value for state i with the highest likelihood. Hence, the most likely estimate of the function value for state i is the sum of the approximation of each matching classifier, weighted by the normalised inverse variance of that classifier.

When taking a closer look at the value model variance $\phi(i)'\Sigma_k^w\phi(i) + \Xi_k$, we can see that it consists of the variance caused by the uncertainty of the value estimate by the classifier weight, $\phi(i)'\Sigma_w^w\phi(i)$, and the measurement noise variance Ξ_k. The uncertainty of the estimate decreases with the number of observations, which causes the value estimate to be judged as being more certain and have a higher influence, the more observations we have made. The noise variance should be independent of the number of observations and gives a bias to the value model variance. Therefore, combining both adjusts classifier



(a) (b)

Figure 3: Applying different mixing parameter ν to approximating a (a) sinusoid in range $[0, \pi)$, and (b) a step function $g(i)$.

mixing by i) the time and state-dependent certainty of the classifier's approximation and ii) the final quality of its approximation.

An underlying assumption of our model is that the measurement noise can be modelled by a Gaussian. In our case, the measurement noise represents the deviation of the function values of V from our linear architecture, about the distribution of which we cannot make any predictions. In the value model variance, the Gaussian nature is partially expressed through the state-dependent term $\phi(i)'\Sigma_k^w\phi$. We currently cannot give any qualitative comments about the influence of this term on the overall estimate, but assume that it is of advantage to functions where the deviation is indeed expressible by a Gaussian, and of possible disadvantage to functions where that is not the case. However, this statement is speculative and definitely needs further investigation.

For now, let us drop the term $\phi(i)'\Sigma_k^w\phi$ from the value model variance, and let it be defined by $\sigma_k = \sqrt{\Xi_k}$. Then the above expression for the most likely value estimate becomes

$$\tilde{V}(i) = \sum_{k=1}^{K} w_k'\phi(i)\frac{I_{S_k}(i)\Xi_k^{-1}}{\sum_{p=1}^{K} I_{S_p}(i)\Xi_p^{-1}}.$$

By noting that ε_k is the approximation for Ξ_k, and by combining Eq. (5) and Eq. (17), we can see that the above is equivalent to using classifier mixing as in Eq. (17) with $\nu = 1$. Thus, using $\nu = 1$ gives the MLE of the value estimate, given that the value variance is approximated by the approximation error ε_k.

5.3 Investigating Parameter ν

From the previous section we could assume that setting $\nu = 1$ gives the best overall approximation. In this section we will show that this might indeed be a good value, but that setting ν is actually not that clear-cut.

By its definition (Eq. (17)), ν influences the spread in weighting between classifiers with different accuracy. The higher ν, the more importance is given to a low approximation error. In the limiting case, which is $\nu \to \infty$, the mixing weights are set that only the approximation of the seemingly most accurate classifier is considered, that is

$$\lim_{\nu\to\infty} \psi_k(i) = \lim_{\nu\to\infty} \frac{I_{S_k}(i)\varepsilon_k^{-\nu}}{\sum_{p=1}^{K} I_{S_p}(i)\varepsilon_p^{-\nu}} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_{p\in\{1,\dots,K\}} \varepsilon_p \\ 0 & \text{otherwise.} \end{cases}$$

Setting it to 0 weights all matching classifiers equally, independent of their accuracy.

Intuitively, setting $\nu = \infty$ only considers the classifier with the lowest error and should therefore give the best approximation. This error, however, is the *average* error over the classifier's match set and does not tell us anything about the approximation error for particular states. Hence, in the worst case, the classifier could happen to only have the lowest error within a set of states for which its approximation is actually worse than that of other classifiers. In such a case, more moderate settings for ν might be appropriate.

Let us consider two simple examples, as shown in Figure 3: In the first example we want to approximate $V(i) =\in (i)$ over the range $[0, \pi)$, using 3 classifiers with match sets $S_1 = [0, \frac{\pi}{2})$, $S_2 = [\frac{\pi}{2}, \pi)$, and $S_3 = [0, \pi)$. Its approximation for $\nu = 0$, $\nu = 1$ and $\nu = \infty$ is shown in Figure 3(a). The second example concerns the step-function

$$g(i) = \begin{cases} 1 & \text{if } i < 0.5, \\ 0 & \text{otherwise,} \end{cases}$$

which we approximate by another 3 classifiers which match $S_1 = [0, 0.5)$, $S_2 = [0.5, 1)$, and $S_3 = [0, 1)$ respectively. Its mixed approximation is shown in Figure 3(b) for the mixing parameters $\nu = 0$ and $\nu = \infty$. Both graphs show that $\nu = 0$ might never be a good idea, as it ignores the approximation error of the classifiers. The case is not as clear-cut for values $\nu > 0$, particularly around $i = \frac{\pi}{2}$ for the sinusoid, where some value $0 < \nu < 1$ might give the best approximation. In the case of the step-function, classifiers 1 and 2 are completely accurate ($\varepsilon_1 = \varepsilon_2 = 0$) and therefore dominate the mixing for any $\nu > 0$. This is an extreme case and might not happen frequently, but is always appropriate, as zero-error classifiers do not have any approximation error for any of its matching states.

To summarise, $\nu = 0$ ignores the accuracy of a classifier and might therefore lead to rather inaccurate overall approximations. On the other hand, $\nu = \infty$ might overemphasise low-error classifiers in certain cases and increase the discontinuity in the approximation. In general, the topic requires further investigation, but to our current knowledge, $\nu = 1$ actually seems to give the best compromise between the two extremes.

6 Summary and Conclusion

In this paper, we have introduced parts of a formal framework that aims at studying learning classifier systems, concerning how a set of classifiers approximate a function separately and in combination. The notation and structure of the framework is aligned to allow for relation of the separate components of LCS to common Machine Learning techniques, and is inspired by the one presented in [6].

The framework is already a contribution by itself, as it clearly reveals the structure of the function approximation architecture independent of, but related to the other components of LCS. Furthermore, it identifies the approximation goal of a classifier as minimising the mean-squared approximation error over all states that it matches, and defines what that goal means for sample-based approximation, together with a proof for the appropriateness of that interpretation. The usefulness of the framework was firstly demonstrated in the discussion about partially matching classifiers, where we have given another interpretation of matching by itself, and that partial matching is mathematically equal to a modification of the sampling probability of the partially matched states, which affects the approximation goal. In addition, we have informally shown how the mixing policy of aggregating classifier approximations rather than averaging over them might compromise the stability of an accuracy-based LCS.

The section on the optimality of a classifier's approximation takes the previously discussed approximation goals further and turns them into formal conditions for both the full-knowledge and the sample-based case, which relates classifier approximation to the Principle of Orthogonality, and also reveals the minimum error of particular classifiers. The measure of a minimum error is significant as it gives the lower bound on the approximation error, and therefore determines the quality of a classifier. For averaging classifiers we have demonstrated that this minimum error is equal to the normalised sample variance over all matched states. For state spaces of countable size we have additionally demonstrated that an optimal approximation is equivalent to orthogonally projecting the function vector into the approximation space, which gives the minimum error by the distance between the function vector and its approximation and again conforms to the Principle of Orthogonality.

A further demonstration of the framework's usefulness is the derivation of algorithmic approaches towards optimal approximations from first principles. Initially, we have derived the LMS algorithm as a local approximation to steepest gradient descent, and have discussed its sensitivity to the chosen step size and the input range. Additionally, we have introduced the Kalman filter by performing Bayesian updates on a normally distributed estimate of the optimal approximation, and have derived forms that allow for unbiased estimations. Due to its equivalence to a form of the Recursive Least Squares algorithm we have shown that our implementation is optimal in the least-squared error sense, and that it obeys the Principle of Orthogonality when applied in conformance to the Minimum Model Error philosophy. Additionally, we have introduced a form of error tracking conforming to the same philosophy that is superior to all currently used methods, and have demonstrated that superiority in experiments that compare the Kalman filter to currently common implementations.

With respect to mixing classifiers we have separated the classifier quality measure used for mixing from the one that acts as a fitness measure in classifier replacement to provide a simplified and more transparent mixing equation. It is shown that the mixing equation is directly related to the Maximum Likelihood Estimate when using the estimation model that was previously developed for the Kalman filter. In the process of evaluating that estimate, we have shown that the development of the Kalman filter has given rise to another possible classifier quality measure that incorporates the approximation uncertainties in addition to the baseline approximation error and might give more stable mixing.

Overall, even though only a part of the complete framework was introduced, it has already proven to be extremely useful in analysing current practices and deploying new methods in LCS. Work on how to employ that framework to relate reinforcement learning to LCS function approximation has already been performed and can be found in [15]. Future work will include investigations on the question of classifier replacement and how that influences the function approximation process, and reinforcement learning and function approximation in combination. We expect that, on completion of the framework, it will be possible to design and create new LCS architectures that demonstrate considerable performance improvements, increased stability, and greater control of parameters. This will greatly improve LCS for industrial applications, extending the already class-leading performance in Data-Mining, and enabling application to more complex delayed-reward problems.

A Derivations and Proofs

A.1 Deriving the Kalman Filter Bayesian Update Equations

We aim at deriving the expectations, covariances, and variances, as required by the Bayesian update

$$\begin{aligned}\mathbb{E}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t})) &= \mathbb{E}(W_{k,t-1}) + \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}(V(i_t) - \mathbb{E}(V_t)),\\ \mathrm{cov}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t})) &= \mathrm{cov}(W_{k,t-1}) - \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}\mathrm{cov}(V_t, W_{k,t-1}).\end{aligned}$$

By definition we have

$$\begin{aligned}\mathbb{E}(W_{k,t}) &= w_{k,t},\\ \mathrm{cov}(W_{k,t}) = \mathrm{cov}(W_{k,t}, W_{k,t}) &= \Sigma^w_{k,t}.\end{aligned}$$

From the Kalman system model (Eq. (11)), by replacing W_k by its a-priori estimate $W_{k,t-1}$, we can derive

$$\begin{aligned}\mathbb{E}(V_t) &= \mathbb{E}(W'_{k,t-1}\phi(i_t)) + \mathbb{E}(\xi_{k,t}) = w'_{k,t-1}\phi(i_t),\\ \mathrm{var}(V_t) &= \mathbb{E}((V_t - \mathbb{E}V_t)^2)\\ &= \mathbb{E}((W'_{k,t-1}\phi(i_t) + \xi_{k,t} - w'_{k,t-1}\phi(i_t))^2)\\ &= \mathbb{E}(((W_{k,t-1} - \mathbb{E}W_{k,t-1})'\phi(i_t) + \xi_{k,t})^2)\\ &= \phi(i_t)'\mathbb{E}((W_{k,t-1} - \mathbb{E}W_{k,t-1})(W_{k,t-1} - \mathbb{E}W_{k,t-1})')\phi(i_t) + \mathbb{E}(\xi^2_{k,t})\\ &= \phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t}.\end{aligned}$$

Equally, we can derive the covariance between $W_{k,t-1}$ and V_t:

$$\begin{aligned}\mathrm{cov}(W_{k,t-1}, V_t) &= \mathbb{E}((W_{k,t-1} - \mathbb{E}W_{k,t-1})(V_t - \mathbb{E}V_t)')\\ &= \mathbb{E}((W_{k,t-1} - \mathbb{E}W_{k,t-1})(\phi(i_t)'W_{k,t-1} + \xi_{k,t} - \phi(i_t)'\mathbb{E}W_{k,t-1})')\\ &= \mathbb{E}((W_{k,t-1} - \mathbb{E}W_{k,t-1})(W_{k,t-1} - \mathbb{E}W_{k,t-1})') + \mathbb{E}(W_{k,t-1} - \mathbb{E}W_{k,t-1})\mathbb{E}(\xi_{k,t})\\ &= \Sigma^w_{k,t-1}\phi(i_t),\\ \mathrm{cov}(V_t, W_{k,t-1}) &= \phi(i_t)'\Sigma^w_{k,t-1}.\end{aligned}$$

To only perform an update for states that classifier k matches, we replace $\phi(i_t)$ by $\sqrt{I_{S_k}(i_t)}\phi(i_t)$ and $V(i_t)$ by $\sqrt{I_{S_k}(i_t)}V(i_t)$. This change is justified by observing that including the function I_{S_k} in the state error $I_{S_k}(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t))^2$ causes the same modifications. The effect is that non-matching states seem to have zero error, as $\sqrt{I_{S_k}(i_t)}\phi(i_t) = \sqrt{I_{S_k}(i_t)}V(i_t) = 0$.

Substituting for the expectations, variances, and covariances gives

$$\begin{aligned}w_{k,t} &= \mathbb{E}(W_{k,t}) = \mathbb{E}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t}))\\ &= \mathbb{E}(W_{k,t-1}) + \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}(V(i_t) - \mathbb{E}(V_t))\\ &= w_{k,t-1} + \Sigma^w_{k,t-1}\sqrt{I_{S_k}(i_t)}\phi(i_t)\left(\sqrt{I_{S_k}(i_t)}\phi(i_t)'\Sigma^w_{k,t-1}\sqrt{I_{S_k}(i_t)}\phi(i_t) + \Xi_{k,t}\right)^{-1}\\ &\qquad\times\left(\sqrt{I_{S_k}(i_t)}V(i_t) - \sqrt{I_{S_k}(i_t)}w'_{k,t-1}\phi(i_t)\right)\\ &= w_{k,t-1} + I_{S_k}(i_t)\Sigma^w_{k,t-1}\phi(i_t)\left(I_{S_k}(i_t)\phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t}\right)^{-1}\left(V(i_t) - w_{k,t-1}\phi(i_t)\right),\end{aligned}$$

and

$$\begin{aligned}\Sigma^w_{k,t} &= \mathrm{cov}(W_{k,t}, W_{k,t}) = \mathrm{cov}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t}))\\ &= \mathrm{cov}(W_{k,t-1}, W_{k,t-1}) - \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}\mathrm{cov}(V_t, W_{k,t-1})\\ &= \Sigma^w_{k,t-1} - \Sigma^w_{k,t-1}\sqrt{I_{S_k}(i_t)}\phi(i_t)\left(\sqrt{I_{S_k}(i_t)}\phi(i_t)'\Sigma^w_{k,t-1}\sqrt{I_{S_k}(i_t)}\phi(i_t) + \Xi_{k,t}\right)^{-1}\\ &\qquad\times\sqrt{I_{S_k}(i_t)}\phi(i_t)'\Sigma^w_{k,t-1}\\ &= \Sigma^w_{k,t-1} - I_{S_k}(i_t)\Sigma^w_{k,t-1}\phi(i_t)\left(I_{S_k}(i_t)\phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t}\right)^{-1}\phi(i_t)'\Sigma^w_{k,t-1},\end{aligned}$$

which finalises the Kalman filter update equations.

A.2 Deriving the Inverse Covariance Form

For this derivation we will apply the Matrix Inversion Lemma, as given in [20, Ch. 9.2]:

Let A and B be two positive-definite $M \times M$ matrices related by

$$A = B^{-1} + CD^{-1}C',$$

where D is a positive-definite $N \times M$ matrix and C is an $M \times N$ matrix. According to the Matrix Inversion Lemma, we may express the inverse of matrix A as

$$A^{-1} = B - BC(C'BC + D)^{-1}C'B.$$

By combining Eq. (12) and Eq. (14), we get the covariance update

$$\Sigma^w_{k,t} = \Sigma^w_{k,t-1} - I_{S_k}(i_t)\Sigma^w_{k,t-1}\phi(i_t)(I_{S_k}(i_t)\phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t})^{-1}\phi(i_t)'\Sigma^w_{k,t-1}.$$

Note that if we set $A^{-1} = \Sigma^w_{k,t}$, $B = \Sigma^w_{k,t-1}$, $C = \sqrt{I_{S_k}(i_t)}\phi(i_t)$, and $D = \Xi_{k,t}$, the update equation matches the second part of the Matrix Inversion Lemma. Hence, as $\Sigma^w_{k,t}$ and $\Xi_{k,t}$ are positive-definite, we can derive the covariance update equation in its inverse covariance form, that is

$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)'.$$

The weight update requires some more steps: Firstly, by combining Eq. (12) and (14) we can relate the Kalman gain to the updated covariance by

$$\zeta_{k,t} = I_{S_k}(i_t)\Sigma^w_{k,t}\phi(i_t)\Xi^{-1}_{k,t}.$$

Together with Eq. (13) that gives

$$w_{k,t} = w_{k,t-1} - I_{S_k}(i_t)\Sigma^w_{k,t}\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)'w_{k,t-1} + I_{S_k}(i_t)\Sigma^w_{k,t}\phi(i_t)\Xi^{-1}_{k,t}V(i_t).$$

Pre-multiplying the above by the inverse covariance $(\Sigma^w_{k,t})^{-1}$ and substituting the derived inverse covariance update for the first $(\Sigma^w_{k,t})^{-1}$ on the right-hand side gives the final weight update

$$(\Sigma^w_{k,t})^{-1}w_{k,t} = (\Sigma^w_{k,t-1})^{-1}w_{k,t-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t}V(i_t).$$

A.3 The Recursive Least Squares Algorithm

In this section we will proof Theorem 4.1, based on the proofs in [20, Ch. 9] and [6, Ch. 3].

Firstly we will derive the iterative weight update. Minimising the error at time t,

$$\sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}(V(i_m) - w'_{k,t}\phi(i_m))^2$$

by settings its first derivative to 0 gives the optimal weight vector

$$\left(\sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}\phi(i_m)\phi(i_m)'\right) w_k = \sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}V(i_m)\phi(i_m).$$

Note that, from the inverse covariance update (Eq. 15) and the fact that $\Sigma^w_{k,-1} = 0$ we can write

$$\begin{aligned}(\Sigma^w_{k,t})^{-1} &= (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)\phi(i_t)' \\ &= \sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}\phi(i_m)\phi(i_m)'.\end{aligned}$$

Using all of above lets us derive

$$\begin{aligned}(\Sigma^w_{k,t})^{-1}w_{k,t} &= I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t) + \sum_{m=0}^{t-1} I_{S_k}(i_m)\Xi^{-1}_{k,m}V(i_m)\phi(i_m) \\ &= I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t) + (\Sigma^w_{k,t-1})^{-1}w_{k,t-1} \\ &= I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t) + (\Sigma^w_{k,t})^{-1}w_{k,t-1} - I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)\phi(i_t)'w_{k,t-1} \\ &= (\Sigma^w_{k,t})^{-1}w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t)).\end{aligned}$$

Pre-multiplying the above by $\Sigma^w_{k,t}$ gives the final weight update

$$w_{k,t} = w_{k,t-1} + I_{S_k}\Xi^{-1}_{k,t}\left((\Sigma^w_{k,t})^{-1}\right)^{-1}\phi(i_t)\left(V(i_t) - w'_{k,t-1}\phi(i_t)\right).$$

The second part of the theorem treats the case of a time-invariant measurement noise variance, that is $\Xi_{k,t} = \Xi_k$, for all $t = 0, 1, \ldots$. This lets us rewrite the error function as

$$\Xi_k^{-1}\sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{k,t}\phi(i_m))^2,$$

which clearly shows that the noise variance becomes a constant factor scaling the error and can be omitted when minimising this function. From this follows that we get the same sequence of weights for any Ξ_k, which is why we can omit it from the update equations (which is equal to setting $\Xi_k = 1$). As above error is a scaled version of the approximated mean squared error (Eq. 4), and the RLS update tracks the weight vector that minimises this error, the Principle of Orthogonality applies (see Theorem 3.2), from which we can also adapt the simplified error function.

Before deriving the error update, let us first derive the useful equality

$$\begin{aligned}
\sum_{m=0}^{t} I_{S_k}(i_m)(w'_k\phi(i_m))^2 &= w'_{k,t}\left(\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)'\right)w_{k,t} \\
&= w'_{k,t}(\Sigma^w_{k,t})^{-1}w_{k,t} \\
&= w'_{k,t}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m).
\end{aligned}$$

Together with $\varepsilon_{k,t}$ from Theorem 3.2, and the RLS weight update with $\Xi_k = 1$, we get

$$\begin{aligned}
(c_{k,t}-1)\varepsilon_{k,t} &= \sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)^2 - \sum_{m=0}^{t} I_{S_k}(w'_{k,t}\phi(i_m))^2 \\
&= \sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)^2 - w'_{k,t}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m) \\
&= \sum_{m=0}^{t-1} I_{S_k}(i_m)V(i_m)^2 + I_{S_k}(i_t)V(i_t)^2 \\
&\quad - I_{S_k}(i_t)w'_{k,t-1}V(i_t)\phi(i_t) - w'_{k,t-1}\sum_{m=0}^{t-1} I_{S_k}(i_m)V(i_m)\phi(i_m) \\
&\quad - I_{S_k}(i_t)\phi(i_t)'\Sigma^w_{k,t}(V(i_t) - w'_{k,t-1}\phi(i_t))\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m).
\end{aligned}$$

Using

$$\begin{aligned}
(c_{k,t-1}-1)\varepsilon_{k,t-1} &= \sum_{m=0}^{t-1} I_{S_k}(i_m)V(i_m)^2 - w'_{k,t-1}\sum_{m=0}^{t-1} I_{S_k}(i_m)V(i_m)\phi(i_m), \\
w'_{k,t}\phi(i_t) &= \phi(i_t)'\Sigma^w_{k,t}\sum_{m=0}^{t} I_{S_k}(i_t)V(i_t)\phi(i_t),
\end{aligned}$$

gives for above

$$\begin{aligned}
(c_{k,t}-1)\varepsilon_{k,t} &= (c_{k,t-1}-1)\varepsilon_{k,t-1} + I_{S_k}(i_t)V(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t)) \\
&\quad - I_{S_k}(i_t)w'_{k,t}\phi(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t)) \\
&= (c_{k,t-1}-1)\varepsilon_{k,t-1} + I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))(V(i_t) - w'_{k,t-1}\phi(i_t)),
\end{aligned}$$

which completes the proof.

A.4 Optimal Approximation for a Sinusoid

Let us consider the function $V(i) = \sin(i)$ and feature vectors $\phi(i) = (1, i)'$, which are used by classifier k, matching range $S_k = [a, b)$ with $b > a$, to approximate function V by uniform sampling. According to Theorem 3.1, the optimal weight vector can be calculated by $w_k = \mathbb{E}_l(\phi\phi')^{-1}\mathbb{E}_k(V\phi)$. Hence, we require to know $\mathbb{E}(I_{S_k})$, $\mathbb{E}(I_{S_k}\phi\phi')$ and $\mathbb{E}(I_{S_k}V\phi)$.

As we only sample states from S_k, we have $\mathbb{E}(I_{S_k}) = 1$. For the other two expectations we get

$$\begin{aligned}
\mathbb{E}(I_{S_k}\phi\phi') &= \int_a^b \frac{1}{b-a}\begin{pmatrix} 1 \\ i \end{pmatrix}\begin{pmatrix} 1 & i \end{pmatrix}\mathrm{d}x = \frac{1}{b-a}\begin{pmatrix} b-a & \frac{b^2-a^2}{2} \\ \frac{b^2-a^2}{2} & \frac{b^3-a^3}{3} \end{pmatrix}, \\
\mathbb{E}(I_{S_k}V\phi) &= \int_a^b \frac{1}{b-a}\begin{pmatrix} 1 \\ i \end{pmatrix}\sin(i)\mathrm{d}i = \frac{1}{b-a}\begin{pmatrix} \cos(a) - \cos(b) \\ a\cos(a) - \sin(a) - b\cos(b) + \sin(b) \end{pmatrix}.
\end{aligned}$$

By inverting $\mathbb{E}(I_{S_k}\phi\phi')$ we can calculate w_k, which gives

$$w_k = \frac{2}{(b-a)^3}\begin{pmatrix} 2(a^2+ab+b^2) & -3(a+b) \\ -3(a+b) & 6 \end{pmatrix}\begin{pmatrix} \cos(a) - \cos(b) \\ a\cos(a) - \sin(a) - b\cos(b) + \sin(b) \end{pmatrix}.$$

To get the mean-squared error $f_k(w_k)$, we again apply Theorem 3.1 to get

$$\begin{aligned}
f_k(w_k) &= \mathbb{E}_k(V^2) - w_k'\mathbb{E}_k(\phi\phi')w_k \\
&= \frac{1}{2(b-a)}\left(\cos(a)\sin(a) - a - \cos(b)\sin(b) + b\right) \\
&\qquad -\frac{w_k'}{b-a}\begin{pmatrix} \cos(a) - \cos(b) \\ a\cos(a) - \sin(a) - b\cos(b) + \sin(b) \end{pmatrix}.
\end{aligned}$$

References

[1] Brian D. O. Anderson and John B. Moore. *Optimal Filtering*. Information and System Sciences Series. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

[2] Alwyn Barry. Limits in long path learning with XCS. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1832–1843. Springer-Verlag, 2003.

[3] Alwyn Barry, John Holmes, and Xavier Llora. Data Mining using Learning Classifier Systems. In Larry Bull, editor, *Foundations of Learning Classifier Systems*, Berlin, 2004. Springer Verlag.

[4] Alwyn M. Barry. The stability of long action chains in XCS. *Journal of Soft Computing*, 6(3–4):183–199, 2002.

[5] Ester Bernadó, Xavier Llorà, and Josep M. Garrell. XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks. In *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS-2001)*, pages 337–341, 2001.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[7] H.-G. Beyer, U.-M. O'Reilly, D.V. Arnold, W. Banzhaf, C. Blum, E.W. Bonabeau, E. Cant Paz, D. Dasgupta, K. Deb, J.A. Foste r, E.D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G.R. Raidl, T. Soule, A. Tyrrell, J.-P. Watson, and E. Zitzler, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005*, volume 2, New York, 2005. ACM Press.

[8] Larry Bull. On accuracy-based fitness. *Journal of Soft Computing*, 6(3–4):154–161, 2002.

[9] Larry Bull. Two Simple Learning Classifier Systems. In Bull and Kovacs [10].

[10] Larry Bull and Tim Kovacs, editors. *Foundations of Learning Classifier Systems*, volume 183 of *Studies in Fuzziness and Soft Computing*. Springer Verlag, Berlin, 2005.

[11] Martin Butz. Kernel-based, Ellipsoidal Conditions in the Real-Valued XCS Classifier System. In Beyer et al. [7], pages 1835–1842.

[12] Martin Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 2004.

[13] Phillip William Dixon, David W. Corne, and Martin John Oates. A preliminary investigation of modified XCS as a generic data mining tool. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 2321 of *LNAI*, pages 133–150. Springer-Verlag, Berlin, 2002.

[14] Scott C. Douglas. A Family of Normalized LMS Algorithms. *IEEE Signal Processing Letters*, SPL-1(3):49–51, March 1994.

[15] Jan Drugowitsch and Alwyn M. Barry. A Formal Framework for Reinforcement Learning with Function Approximation in Learning Classifier Systems. Technical Report 2006–02, University of Bath, U.K., January 2006.

[16] Eweda Eweda and Odile Macchi. Convergence of the RLS and LMS Adaptive Filter. *IEEE Transactions on Circuits and Systems*, CAS-34(7):799–803, July 1987.

[17] Andrew Greenyer. The use of a learning classifier system JXCS. In P. van der Putten and M. van Someren, editors, *CoIL Challenge 2000: The Insurance Company Case*. Leiden Institute of Advanced Computer Science, June 2000. Technical report 2000-09.

[18] Geoffrey Grimmett and David Stirzaker. *Probability and Random Processes*. Oxford University Press, New York, 3rd edition, 2001.

[19] Simon Haykin. *Neural Networks. A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1999.

[20] Simon Haykin. *Adaptive Filter Theory*. Information and System Sciences Series. Prentice Hall, Upper Saddle River, NJ, 4th edition, 2002.

[21] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[22] Rudolph Emil Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions ASME, Part D (J. Basic Engineering)*, 83:95–108, 1961.

[23] Tim Kovacs. *A Comparison and Strength and Accuracy-based Fitness in Learning Classifier Systems*. PhD thesis, University of Birmingham, 2002.

[24] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Extending XCSF Beyond Linear Approximation. In Beyer et al. [7], pages 1827–1834.

[25] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Generalization in the XCSF Classifier Systems: Analysis, Improvement, and Extenstion. Technical Report 2005012, Illinois Genetic Algorithms Laboratory, March 2005.

[26] Peter S. Maybeck. *Stochastic Models, Estimation, and Control. Volume 1*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, Inc., New York, 1979.

[27] D. J. Mook and J. L. Junkins. Minimum Model Error Estimation for Poorly Modeled Dynamic Systems. *Journal of Guidance, Control and Dynamics*, 11(3):256–261, May-June 1988.

[28] Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *LNAI*, pages 223–242, Berlin, 2000. Springer-Verlag.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. A Bradford Book.

[30] John Tsitsiklis and Benjamin Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.

[31] Atsushi Wada, Keiki Takadama, Katsunori Shimohara, and Osamu Katai. Learning Classifier System with Convergence and Generalisation. In Bull and Kovacs [10].

[32] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report TR 95-401, University of North Carolina at Chapel Hill, Department of Computer Science, April 2004.

[33] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Revord Part IV*, pages 96–104, 1960.

[34] Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. http://prediction-dynamics.com/.

[35] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[36] Stewart W. Wilson. Function Approximation with a Classifier System. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981. Morgan Kaufmann, 2001.

[37] Stewart W. Wilson. Classifiers that Approximate Functions. *Neural Computing*, 1(2-3):211–234, 2002.

[38] Stewart W. Wilson. Classifier Systems for Continuous Payoff Environments. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004*, volume 3103 of *Lecture Notes in Computer Science*, pages 824–835. Springer Verlag, 2004.